Unterschrift des Betreuers

TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

DIPLOMARBEIT

Boltzmann Sampling Algorithmen zum Erzeugen zufälliger kombinatorischer Objekte

Ausgeführt am Institut für
Diskrete Mathematik und Geometrie
der Technischen Universität Wien

unter der Anleitung von
Ao. Univ. Prof. Dipl.Ing. Dr. techn. Alois Panholzer

durch
Stefan Schnabl

Oskar-Helmer-Straße 61/9
3100 St. Pölten

Datum

Unterschrift

Kurzfassung

In dieser Arbeit betrachten wir die sogenannten *Boltzmann-Sampler*. Hierbei handelt es sich um Generatoren von zufälligen kombinatorischen Objekten. Welcher Art diese Objekte sind, wird hierbei durch eine kombinatorische Spezifikation bestimmt, also ein Gleichungssystem, das Terme aus kombinatorischen Basisoperationen in Relation setzt.

Die Boltzmann-Sampler wurden in einem Artikel von Duchon, Flajolet, Louchard und Schaeffer im Jahr 2002 eingeführt. Im Gegensatz zu früher entwickelten Methoden zur Erzeugung zufälliger Strukturen hat der Boltzmann-Sampler den Vorteil, dass bei Fallenlassen der Einschränkung, dass die Größe der Strukturen fest vorgegeben ist, sondern in einem Intervall liegt, in vielen Fällen eine sehr effiziente Erzeugung der Strukturen in $O(n)$ möglich ist, wobei n die Strukturgröße bezeichnet.

In dieser Arbeit formulieren wir das Boltzmann-Sampling im Kontext der sogenannten *kombinatorischen Spezies*, einer relativ neuen Theorie, die durch Joyal 1981 begründet wurde. In dieser Theorie ist es möglich, sowohl markierte, als auch unmarkierte kombinatorische Objekte in einem Zug zu betrachten. Weiters wurde die Frage, wann ein kombinatorisches Gleichungssystem sinnvoll ist und wie die Lösung zu konstruieren ist, befriedigend geklärt. Der erste Teil dieser Arbeit ist eine Einführung in die Theorie der kombinatorischen Spezies inklusive der entwickelten Sätze über die kombinatorischen Gleichungssysteme.

Im zweiten Abschnitt werden die exponentiellen Boltzmann-Sampler im Kontext der Spezies-Theorie definiert und untersucht. Diese Sampler erzeugen markierte Objekte, deren Grundbausteine mit einer eindeutigen Marke versehen sind. Es werden Sampler für die wichtigsten Basisklassen von kombinatorischen Objekten und die wichtigsten kombinatorischen Basisoperationen angegeben, sowie deren Korrektheit bewiesen. Außerdem wird die *kombinatorische Newton-Iteration* vorgestellt, die eine effiziente Auswertung der erzeugenden Funktion, die für den Sampling-Prozess benötigt wird, ermöglicht.

Im letzten Abschnitt wird die Implementierung eines exponentiellen Boltzmann-Samplers in der Programmiersprache *Java* beschrieben, die im Zuge dieser Arbeit erstellt wurde. Das Programm erhält dabei als Eingabedaten eine kombinatorische Spezifikation, die in einem *XML*-Format angegeben wird, sowie den gewünschten Größenbereich der zu erzeugenden Strukturen, und sampelt diese. Diese Implementierung wurde absichtlich so allgemein und erweiterbar wie möglich gehalten, damit sie für unterschiedlichste Anwendungsbereiche verwendbar ist.

Abstract

This thesis is about *Boltzmann samplers*, which are generators of random combinatorial objects. The structure of these objects is determined by a combinatorial specification, which is a system of equations consisting of terms of combinatorial base operations.

The Boltzmann samplers were introduced in the year 2002 in an article by Duchon, Flajolet, Louchard and Schaeffer. In contrast to previously developed sampling methods, they have the advantage of a very efficient runtime behavior. If the size condition on the sampled objects is loosened to a size interval instead of an exact size, in many cases the runtime is $O(n)$ with respect to structure size n.

In this paper the Boltzmann sampling framework is formulated in the context of the theory of combinatorial species, which was introduced relatively recently by Joyal in 1981. Within this theory it is possible to treat labeled and unlabeled objects simultaneously, contrasting the older combinatorial frameworks. Furthermore it provides satisfying answers to questions concerning the meaning given to a combinatorial systems of equations. The first part of this paper gives an introduction to the theory of species including the answers to these questions.

In the second part we define and examine exponential Boltzmann samplers using the theory of species. These samplers generate objects, whose basic building blocks are marked with unique labels. We state sampling algorithms for the most important base classes of objects and combinatorial base operations and prove their correctness. Finally we present a combinatorial version of *Newton's iteration*, enabling us to efficiently evaluate the generating functions used by the sampling process.

The last section is dedicated to the implementation of an exponential Boltzmann sampler written in the *Java* programming language. The program's input data consist of a combinatorial specification written in the *XML* file format, as well as the desired range of size of the generated structures. The structures get sampled and can be saved to a file in a text format. The implementation is supposed to be as generic and expandable as possible in order to meet the requirements of many application areas.

Danksagung

Ich möchte mich hiermit herzlich bei meinem Diplomarbeitsbetreuer Prof. Alois Panholzer bedanken. Er räumte mir die größtmögliche Freiheit bezüglich der Gestaltung und Durchführung meiner Arbeit ein. Außerdem stand er mir immer spontan für Fragen zur Verfügung. Meinem Arbeitgeber, der Firma GEBOS. m.b.H. möchte ich für die flexible Gestaltungsmöglichkeit meiner Arbeitszeit danken, ohne die mein Studium nicht möglich gewesen wäre. Weiters danke ich meinen Verwandten für ihre Unterstützung während meiner Studienzeit. Abschließend möchte ich meinen Freunden danken, die immer für mich da waren, in welcher Form ich sie auch immer benötigte, seien es die gemeinsamen mathematischen Gespräche, ihre Aufmunterungs- und Motivationsversuche, und vor allem die Zeit abseits der Mathematik, in der sie mein Leben bereichern.

Inhaltsverzeichnis

1 Einleitung

Im Jahr 2002 wurde in einem Artikel von Philippe Duchon, Philippe Flajolet, Guy Louchard und Gilles Schaeffer (siehe [4]) eine Methode zur zufälligen Erzeugung kombinatorischer Objekte namens *Boltzmann-Sampling* eingeführt. Bei kombinatorischen Objekten handelt es sich um Strukturen, die aus Grundbausteinen unter der Anwendung von bestimmten Konstruktionsregeln erzeugt werden.

Diese Methode hat im Vergleich zu den bereits bekannten Verfahren zur zufälligen Erzeugung wie der sogenannten rekursiven Methode [6] oder Markov-Ketten-Methoden (siehe [1]) den Vorteil, dass in vielen praktisch relevanten Fällen eine sehr effiziente Erzeugung von Strukturen möglich ist, also in einem Zeitaufwand, der linear mit der Größe der erzeugten Struktur ansteigt. Dies gilt, wenn dafür die Anforderung fallen gelassen wird, dass die erzeugten Strukturen eine feste Größe haben, sondern wenn man erlaubt, dass die Größe in einem Intervall liegt. Wenn man einen einzelnen festen Größenwert vorgibt, ist es in vielen Fällen trotzdem noch möglich, zumindest in mit der Strukturgröße quadratisch ansteigender Laufzeit zu sampeln. Strukturen einer festen Größe werden durch Boltzmann-Sampler immer mit gleicher Wahrscheinlichkeit erzeugt. Weiters ist es leicht, aus gegebenen Konstruktionsregeln die entsprechenden Boltzmann-Sampler zu erzeugen, da es zu diesen Regeln jeweils korrespondierende Regeln für den Bau eines Samplers gibt.

Um solche Begriffe wie *Struktur*, *Konstruktionsregel* und *Größe* mathematisch exakt zu definieren und verwenden zu können, benötigen wir eine entsprechende Theorie. Die in der Kombinatorik weit verbreitete Theorie der kombinatorischen Klassen, wie sie zum Beispiel in [7] präsentiert wird, und welche in den meisten wissenschaftlichen Arbeiten zum Thema *Boltzmann-Sampling* verwendet wird, hat gewisse Nachteile. Es werden zum Beispiel markierte und unmarkierte Objekte, also solche, bei denen die Grundbausteine eindeutige Marken besitzen, und solche, bei denen die Grundbausteine ununterscheidbar sind, durch zwei getrennte Theorien behandelt. Weiters stellt es ein Problem dar, allgemein zu definieren, welche Konstruktionsregeln zu gültigen kombinatorischen Objekten führen, und welche Regelsammlungen nicht sinnvoll sind.

Im Jahr 1981 begründete Andrè Joyal in seiner Arbeit [9] die Theorie der kombinatorischen Spezies. Hierbei handelt es sich um einen alternativen Ansatz, die obige Problemstellung zu behandeln. Eine kombinatorische Spezies ist ein mathematisches Objekt, das jeder beliebigen endlichen Menge von Grundbausteinen eine endliche Menge von Strukturen zuordnet, und das weiß, wie die Strukturen umgewandelt werden müssen, wenn wir die Grundbausteinmenge bijektiv auf eine andere Menge von Grundbausteinen abbilden. Diese Theorie behandelt markierte und unmarkierte Objekte gleichzeitig, indem ein unmarkiertes Objekt einfach als Äquivalenzklasse von markierten Objekten betrachtet wird, die durch oben beschriebene bijektive Abbildung der Grundbausteinmenge ineinander übergehen.

Weiters existieren für diese Theorie hinreichende Bedingungen für die Sinnhaftigkeit der Konstruktionsregeln kombinatorischer Objekte, die in Form von Gleichungssystemen, in denen Terme aus Operationen an Spezies vorkommen, gegeben sind. In einer Arbeit von Carine Pivoteau, Bruno Salvy und Michèle Soria aus dem Jahr 2012 (siehe [14]) wurden die entsprechenden Sätze präsentiert. Darüber hinaus wird sogar eine kombinatorische Version der aus der Analysis bekannten *Newton-Iteration* vorgestellt, die sehr gute Konvergenzeigenschaften hat. Diese ist für das Boltzmann-Sampling sehr sinnvoll anwendbar, da sie ermöglicht, das sogenannte *Boltzmann-Orakel* zu implementieren. Hierbei handelt es sich um die Möglichkeit, sogenannte erzeugende Funktionen auszuwerten. Dies sind Objekte, die in der abzählenden Kombinatorik die Anzahl von Strukturen bestimmter Größe codieren, und die unter gewissen Voraussetzungen als analytische Funktionen interpretiert werden können.

Aus den soeben beschriebenen Punkten ergibt sich für diese Arbeit folgender Aufbau. Im ersten Kapitel werden die kombinatorischen Grundlagen für das Boltzmann-Sampling entwickelt. Dazu

zählt eine Einführung in die Theorie der kombinatorischen Spezies, eine Einführung in formale Potenzreihen, die die algebraische Grundlage für erzeugende Funktionen sind, sowie ein Überblick über die in [14] vorgestellten Sätze über kombinatorische Systeme.

Das zweite Kapitel befasst sich mit den exponentiellen Boltzmann-Samplern, also Samplern, die markierte Objekte erzeugen. Es gibt auch gewöhnliche Boltzmann-Sampler, die unmarkierte Objekte generieren, allerdings würden diese aufgrund der viel umfangreicheren Problemstellung den Rahmen dieser Arbeit sprengen. Die exponentiellen Boltzmann-Sampling-Algorithmen werden in der Theorie der kombinatorischen Spezies neu formuliert. Weiters werden wir deren Korrektheit beweisen. Hier soll insbesonders darauf hingewiesen werden, dass eine allgemeinere Form von Algorithmen behandelt wird, als die, die in den Artikeln, die auf der Theorie der kombinatorischen Klassen beruhen, zu finden sind. Ein Unterabschnitt ist der Bestimmung der Größe der zu sampelnden Strukturen gewidmet. Schließlich wird auch noch die bereits erwähnte Newton-Iteration betrachtet.

Der praktische Teil der Arbeit besteht aus der Implementierung eines exponentiellen Boltzmann-Samplers in *Java*. Dieser ist so allgemein gehalten, dass in einer Textdatei eine nahezu beliebige kombinatorische Spezifikation an das Programm übergeben werden kann, zu der dann Strukturen gesampelt werden. Es wurde viel Wert darauf gelegt, dass das Programm einfach erweiterbar wird, und auch möglichst wenig Abhängigkeiten zu Bibliotheken besitzt. Die Module bauen nur auf Interfaces anderer Module, nicht aber auf konkreten Implementierungen dieser Interfaces auf. Daher werden einzelne Subsysteme leicht austauschbar beziehungsweise erweiterbar. Die einzige Bibliothek, die verwendet wird, ist die Java-Klassenbibliothek. Alle anderen benötigten Module wie ein Löser von linearen Gleichungssystemen oder Zufallszahlengeneratoren, die bestimmte Verteilungen realisieren, sind selber ausprogrammiert.

1.1 Notation

In diesem Abschnitt wird die in der gesamten Arbeit verwendete Notation für bestimmte mathematische Objekte eingeführt. Zuerst betrachten wir bestimmte Mengen.

Definition 1.1.1. Die Menge $\mathbb{N}_0$ bezeichnet die Menge der natürlichen Zahlen inklusive 0, also $\{0, 1, 2, \ldots\}$. Die Menge $\mathbb{N}$ bezeichnet die natürlichen Zahlen ohne 0, also $\{1, 2, \ldots\}$. Die Menge $\mathbb{Z}$ bezeichnet die ganzen Zahlen, $\mathbb{Q}$ die rationalen Zahlen, $\mathbb{R}$ die reellen Zahlen und $\mathbb{C}$ die komplexen Zahlen.

Definition 1.1.2. Sei $n \in \mathbb{N}$, so bezeichnet $[n]$ die Menge der natürlichen Zahlen von 1 bis n, also

$$[n] := \big\{k \in \mathbb{N} \,:\, 1 \leq k \leq n\big\}.$$

Nun betrachten wir gewisse immer vorkommende Abbildungen.

Definition 1.1.3. Sei M eine beliebige Menge, so bezeichnen wir die identische Abbildung, die jedes Element der Menge M auf sich selber abbildet, als id_M, es gilt also

$$\mathrm{id}_M : M \to M, x \mapsto x.$$

Wenn die Menge M aus dem Zusammenhang klar ist, schreiben wir auch nur id. Die leere Abbildung, also jene Abbildung, deren Definitionsbereich die leere Menge ist, bezeichnen wir mit dem Symbol $f_\emptyset$.

Definition 1.1.4. Seien A und B zwei Mengen, so bezeichnen wir die Menge aller Bijektionen zwischen A und B mit $\mathrm{Bij}\big(A, B\big)$. Es gilt also

$$\mathrm{Bij}\big(A, B\big) := \big\{f : A \to B \,:\, f \text{ bijektiv}\big\}.$$

Definition 1.1.5. Die Funktion

$$\lfloor \cdot \rfloor : \mathbb{R} \to \mathbb{Z}, x \mapsto \max\{k \in \mathbb{Z} \,:\, k \leq x\}$$

wird als *Floor*-Funktion, die Funktion

$$\lceil \cdot \rceil : \mathbb{R} \to \mathbb{Z}, x \mapsto \min\{k \in \mathbb{Z} \,:\, k \geq x\}$$

als *Ceiling*-Funktion bezeichnet.

Definition 1.1.6. Die sogenannte *Kronecker-Delta-Funktion* ist als

$$\delta_{ij} := \begin{cases} 1 & \text{für } i = j, \\ 0 & \text{sonst,} \end{cases}$$

definiert.

2 Kombinatorische Grundlagen

In diesem Kapitel werden die Grundlagen für die Erzeugung zufälliger kombinatorischer Objekte präsentiert. Die Boltzmann-Sampler, von denen diese Arbeit handelt, erzeugen Zufallsobjekte, die aus einer kombinatorischen Spezifikation hervorgehen.

Die Arbeiten über Boltzmann-Sampler beruhen hauptsächlich auf zwei Theorien. Die erste, in der abzählenden Kombinatorik schon lange etablierte, Theorie ist jene der *kombinatorischen Klassen*. Sie betrachtet unmarkierte und markierte kombinatorische Objekte getrennt und führt jeweils spezifische Operatoren und Typen erzeugender Funktionen zur Abzählung ein. Markierte Objekten setzen sich aus Basisobjekten zusammen, welche mit einer eindeutigen Marke versehen und daher unterscheidbar sind, bei unmarkierten Objekten sind die Atome hingegen ununterscheidbar. Eine detaillierte Einführung ist im Buch von Sedgewick und Flajolet [7] zu finden.

Dieses Kapitel beschäftigt sich mit den *kombinatorischen Spezies*, einer von Joyal 1981 [9] begründeten Theorie, die einen kategorientheoretischen Zugang zu den Problemstellungen der Kombinatorik und zur Behandlung und rigorosen Definition von diskreten Strukturen darstellt. Im Vergleich zur Theorie der kombinatorischen Klassen müssen hier markierte und unmarkierte Objekte nicht getrennt betrachtet werden. Wie wir später sehen werden, gehen aus dieser Theorie Sätze hervor, mit denen die Charakterisierung von wohldefinierten Spezifikationen ermöglicht wird, also solchen, die im Sinne der Kombinatorik sinnvoll sind. Weiters wird in den späteren Kapiteln eine sehr effiziente Methode zur Implementierung des Boltzmann-Orakels zur numerischen Auswertung von erzeugenden Funktionen vorgestellt, die in diesem Framework formuliert ist. Eine umfassende Behandlung der Spezies-Theorie findet man im Buch von Bergeron, Labelle und Leroux [2].

2.1 Formale Potenzreihen

Bevor wir uns mit der Kombinatorik beschäftigen, wenden wir unseren Blick den sogenannten *formalen Potenzreihen* zu. Diese bilden in der modernen Kombinatorik ein mächtiges Werkzeug zur systematischen Anzahlbestimmung von kombinatorischen Objekten.

Den Namen erhalten die formalen Potenzreihen von den klassichen Potenzreihen im Sinne der Analysis, da sie sich die Schreibweise teilen. Im Gegensatz zu den klassischen Potenzreihen handelt es sich bei den formalen allerdings um rein algebraische Objekte. Insbesonders machen Betrachtungen von Konvergenzradien oder ähnlichen Eigenschaften in diesem Kontext keinen Sinn.

Wir werden auf den formalen Potenzreihen eine Topologie definieren. Diese unterscheidet sich allerdings erheblich von den in der Analysis verwendeten Topologien, und definiert eine Art von Konvergenz, die die in der Algebra wichtige Tatsache, dass alle Eigenschaften, die von Interesse sind, in endlich vielen Schritten ermittelt werden können, widerspiegelt.

Der Fall der formalen Potenzreihen in einer bzw. endlich vielen Veränderlichen wird in der Literatur ausgiebig behandelt und ist zum Beispiel in [12, S. 205-212] oder [8] nachzulesen.

In dieser Arbeit werden auch formale Potenzreihen in unendlich vielen Veränderlichen verwendet. Diesen Fall betrachten wir nun näher.

Definition 2.1.1. Sei V eine endliche oder abzählbar unendliche Menge. Ein *Monom auf* V ist eine Abbildung $m : V \to \mathbb{N}_0$, sodass für fast alle $z \in V$ gilt, dass $m(z) = 0$ ist. Enthält $z_{i_1}, \ldots, z_{i_l}$ all jene Elemente von V mit $m(z_{i_k}) \neq 0$ für alle $k \in \{1, \ldots, l\}$ und $n_{i_k} := m(z_{i_k})$, dann schreibt man für das Monom m auch $z_{i_1}^{n_{i_1}} z_{i_2}^{n_{i_2}} \cdots z_{i_l}^{n_{i_l}}$. Das *Nullmonom* ist als $m_0 := (V \to \mathbb{N}_0, v \mapsto 0)$ definiert. Der *Grad* des Monoms m ist als $\deg(m) := \sum_{v \in V} m(v)$ definiert. Da nur endlich viele Summanden ungleich 0 sind, ist diese Definition sinnvoll.

Definition 2.1.2. Sei $(R, +, \cdot)$ ein kommutativer Ring mit Einselement 1, V eine endliche oder abzählbar unendliche Menge und M_V die Menge der Monome auf V. Eine *formale Potenzreihe auf dem Ring* R ist eine Abbildung $f : M_V \to R$. Die Menge der formalen Potenzreihen auf dem Ring R wird als $R[\![V]\!]$ bezeichnet. Besteht die Menge V aus den Elementen $z_1, \ldots, z_n$, so schreiben wir alternativ auch $R[\![z_1, \ldots, z_n]\!]$. Wenn $f \in R[\![V]\!]$ derart beschaffen ist, dass ein Monom $\tilde{m} := z_{i_1}^{n_{i_1}} z_{i_2}^{n_{i_2}} \cdots z_{i_l}^{n_{i_l}}$ existiert, sodass $f(m) = 0$ für alle $m \in M_V \setminus \{\tilde{m}\}$ und $f(\tilde{m}) =: a$ mit $a \neq 0$,

dann schreiben wir für f auch $a \cdot z_{i_1}^{n_{i_1}} z_{i_2}^{n_{i_2}} \cdots z_{i_l}^{n_{i_l}}$, falls $a = 1$ ist auch einfach nur $z_{i_1}^{n_{i_1}} z_{i_2}^{n_{i_2}} \cdots z_{i_l}^{n_{i_l}}$. Sei $z_{i_1}^{n_{i_1}} z_{i_2}^{n_{i_2}} \cdots z_{i_l}^{n_{i_l}} \in M_V$, so wird $f(z_{i_1}^{n_{i_1}} z_{i_2}^{n_{i_2}} \cdots z_{i_l}^{n_{i_l}})$ als *Koeffizient* des Monoms $z_{i_1}^{n_{i_1}} z_{i_2}^{n_{i_2}} \cdots z_{i_l}^{n_{i_l}}$ bezeichnet. Eine andere Schreibweise dafür ist $[z_{i_1}^{n_{i_1}} z_{i_2}^{n_{i_2}} \cdots z_{i_l}^{n_{i_l}}]f$. Meistens werden die Unbestimmten aus der Menge V beim Anschreiben eines Elements von $R[\![V]\!]$ explizit angegeben, wenn dies möglich ist. Sei $f \in R[\![V]\!]$ und $V = \{z_1, \ldots, z_n\}$, so bezeichnen wir die formale Potenzreihe f auch als $f(z_1, \ldots, z_n)$.

Satz 2.1.3. *Sei $(R, +, \cdot)$ ein kommutativer Ring mit Nullelement 0_R und Einselement 1_R, V eine endliche oder abzählbar unendliche Menge und M_V die Menge der Monome auf V, dann bildet die Menge $R[\![V]\!]$ mit den Operationen*

$$\begin{aligned}
+ :&\quad R[\![V]\!] \times R[\![V]\!] \to R[\![V]\!],\ (f,g) \mapsto \big(M_V \to R, m \mapsto f(m) + g(m)\big),\\
- :&\quad R[\![V]\!] \to R[\![V]\!],\ f \mapsto \big(M_V \to R, m \mapsto -f(m)\big),\\
\cdot :&\quad R[\![V]\!] \times R[\![V]\!] \to R[\![V]\!],\ (f,g) \mapsto \Big(M_V \to R, m \mapsto \sum_{\substack{m_1, m_2 \in M_V \\ m_1 + m_2 = m}} f(m_1) \cdot g(m_2)\Big)
\end{aligned}$$

einen kommutativen Ring mit dem Nullelement $0 : M_V \to R, m \mapsto 0$ und dem Einselement

$$1 : M_V \to R, m \mapsto \begin{cases} 1_R & \text{falls } m = m_0, \\ 0_R & \text{sonst.} \end{cases}$$

Beweis. Sei im Folgenden $m \in M_V$ ein beliebiges Monom. Die Assoziativität und Kommutativität der Addition von $R[\![V]\!]$ folgt unmittelbar aus den der Assoziativität und Kommutativität der Addition des Rings R. Für alle $f \in R[\![V]\!]$ gilt

$$(f + 0)(m) = f(m) + 0(m) = f(m) + 0_R = f(m) = 0_R + f(m) = 0(m) + f(m) = (0 + f)(m).$$

0 ist daher das neutrale Element der Addition. Weiters gilt für alle $f \in R[\![V]\!]$, dass

$$(f + (-f))(m) = f(m) + (-f)(m) = f(m) + (-f(m)) = 0_R.$$

Insgesamt ist also $(R[\![V]\!], +, -, 0)$ eine kommutative Gruppe. Für $f, g \in R[\![V]\!]$ gilt

$$\begin{aligned}
(f \cdot g)(m) &= \sum_{\substack{m_1, m_2 \in M_V \\ m_1 + m_2 = m}} f(m_1) \cdot g(m_2) \overset{R \text{ kommutativ}}{=} \sum_{\substack{m_1, m_2 \in M_V \\ m_1 + m_2 = m}} g(m_2) \cdot f(m_1) \\
&= \sum_{\substack{m_1, m_2 \in M_V \\ m_1 + m_2 = m}} g(m_1) \cdot f(m_2) = (g \cdot f)(m).
\end{aligned}$$

Die Multiplikation ist also kommutativ. Dass die Multiplikation assoziativ ist, sehen wir für $f, g, h \in R[\![V]\!]$ folgendermaßen:

$$\begin{aligned}
\big((f \cdot g) \cdot h\big)(m) &= \sum_{\substack{m_1, m_2 \in M_V \\ m_1 + m_2 = m}} (f \cdot g)(m_1) \cdot h(m_2) = \sum_{\substack{m_1, m_2 \in M_V \\ m_1 + m_2 = m}} \Big(\sum_{\substack{m_3, m_4 \in M_V \\ m_3 + m_4 = m_1}} f(m_3) \cdot g(m_4) \Big) \cdot h(m_2) \\
&= \sum_{\substack{m_1, m_2 \in M_V \\ m_1 + m_2 = m}} \sum_{\substack{m_3, m_4 \in M_V \\ m_3 + m_4 = m_1}} f(m_3) \cdot g(m_4) \cdot h(m_2) = \sum_{\substack{m_1, m_2, m_3 \in M_V \\ m_1 + m_2 + m_3 = m}} f(m_1) \cdot g(m_2) \cdot h(m_3) \\
&= \sum_{\substack{m_1, m_2 \in M_V \\ m_1 + m_2 = m}} \sum_{\substack{m_3, m_4 \in M_V \\ m_3 + m_4 = m_2}} f(m_1) \cdot g(m_3) \cdot h(m_4) \\
&= \sum_{\substack{m_1, m_2 \in M_V \\ m_1 + m_2 = m}} f(m_1) \cdot \Big(\sum_{\substack{m_3, m_4 \in M_V \\ m_3 + m_4 = m}} g(m_3) \cdot h(m_4) \Big) = \sum_{\substack{m_1, m_2 \in M_V \\ m_1 + m_2 = m}} f(m_1) \cdot (g \cdot h)(m_2) \\
&= \big(f \cdot (g \cdot h)\big)(m).
\end{aligned}$$

Schließlich zeigen wir noch die Gültigkeit des Distributivgesetzes. Für $f, g, h \in R[\![V]\!]$ gilt

$$\begin{aligned}\big(f \cdot (g+h)\big)(m) &= \sum_{\substack{m_1,m_2 \in M_V \\ m_1+m_2=m}} f(m_1) \cdot (g+h)(m_2) = \sum_{\substack{m_1,m_2 \in M_V \\ m_1+m_2=m}} f(m_1) \cdot \big(g(m_2)+h(m_2)\big) \\ &= \sum_{\substack{m_1,m_2 \in M_V \\ m_1+m_2=m}} \big(f(m_1) \cdot g(m_2)\big) + \big(f(m_1) \cdot h(m_2)\big) \\ &= \sum_{\substack{m_1,m_2 \in M_V \\ m_1+m_2=m}} f(m_1) \cdot g(m_2) + \sum_{\substack{m_1,m_2 \in M_V \\ m_1+m_2=m}} f(m_1) \cdot h(m_2) \\ &= (f \cdot g)(m) + (f \cdot h)(m) = \big((f \cdot g) + (f \cdot h)\big)(m).\end{aligned}$$

Analog kann man zeigen, dass $(f+g) \cdot h = (f \cdot h) + (g \cdot h)$. Es handelt sich bei $R[\![V]\!]$ also um einen kommutativen Ring. Dass die 1 das Einselement ist, sehen wir mittels

$$(f \cdot 1)(m) = \sum_{\substack{m_1,m_2 \in M_V \\ m_1+m_2=m}} f(m_1) \cdot 1(m_2) = f(m) \cdot 1_R = f(m),$$

wenn wir beachten, dass $1(m)$ genau dann 1_R ist, wenn m das Nullmonom ist, und ansonsten 0 ist. □

Bemerkung. Der Ring R erlaubt eine monomorphe Einbettung in $R[\![V]\!]$ derart, dass ein Ringelement r mit der formalen Potenzreihe

$$M_V \to R, m \mapsto \begin{cases} r & \text{falls } m = m_0, \\ 0 & \text{sonst} \end{cases}$$

identifiziert wird. In Zukunft werden wir uns den Ring R daher immer derart in $R[\![V]\!]$ eingebettet denken.

Zusätzlich zu den normalen Ringoperationen benötigen wir im Zuge dieser Arbeit auch die Operation der formalen (partiellen) Ableitung.

Definition 2.1.4. Sei V eine endliche oder abzählbar unendliche Menge, M_V die Menge der Monome auf V und $R[\![V]\!]$ der Ring der formalen Potenzreihen. Sei weiters $z \in V$, dann heißt die als

$$\frac{\partial}{\partial z} : R[\![V]\!] \to R[\![V]\!], f \mapsto f_z$$

definierte Operation die *partielle Ableitung nach* z. Dabei gilt für f_z, dass

$$f_z(m) := m_z(z) \cdot f(m_z) = \underbrace{f(m_z) + \ldots + f(m_z)}_{m_z(z) \text{ mal}} \text{ mit}$$

$$m_z : V \to \mathbb{N}_0, v \mapsto \begin{cases} m(v)+1 & \text{falls } v = z, \\ m(v) & \text{sonst}, \end{cases}$$

für alle $m \in M_V$. Besteht die Menge der Unbestimmten aus genau einem Element z, so nennen wir die Operation *Ableitung*, und schreiben

$$\frac{d}{dz} := \frac{\partial}{\partial z} \text{ und } \frac{d}{dz} f =: f' \text{ für alle } f \in R[\![V]\!].$$

Neben der algebraischen Struktur besitzt $R[\![V]\!]$ auch eine topologische Struktur. Hierbei wird Wert auf ein möglichst einfaches Konvergenzverhalten gelegt, das so aussieht, dass Folgen von formalen Potenzreihen genau dann konvergieren, wenn sich für jedes Monom die Folge der Koeffizienten dieses Monoms schließlich stabilisiert, also schließlich konstant ist.

Definition 2.1.5. Sei R ein kommutativer Ring mit Einselement, V eine endliche oder abzählbar unendliche Menge, M_V die Menge der Monome auf V und $R[\![V]\!]$ der Ring der formalen Potenzreihen. Sei weiters $\mathcal{D}_R := \mathbf{2}^R$ die diskrete Topologie auf R. Wir identifizieren die formalen Potenzreihen aus $R[\![V]\!]$ mittels der Abbildung

$$\iota : R[\![V]\!] \to \prod_{m \in M_V} R, f \mapsto \big(f(m)\big)_{m \in M_V}$$

mit den Elementen des topologischen Raum $\big(\prod_{m\in M_V} R, \prod_{m\in M_V} \mathcal{D}_R\big)$, wobei $\prod_{m\in M_V} \mathcal{D}_R$ die Produkttopologie bezeichnet. Mit der Topologie $\mathcal{T} := \iota^{-1}\big(\prod_{m\in M_V} \mathcal{D}_R\big)$ wird nun $R[\![V]\!]$ zu einem topologischen Raum. Wir werden uns den Ring der formalen Potenzreihen immer mit der Topologie $\mathcal{T}$ versehen denken.

Lemma 2.1.6. *Für jede formale Potenzreihe $f \in R[\![V]\!]$ ist die Menge*

$$\mathcal{B}(f) := \big\{\{g \in R[\![V]\!] \,:\, f(m) = g(m) \text{ für alle } m \in I\} \,:\, I \subseteq M_V, I \text{ endlich}\big\}$$

eine Umgebungsbasis von f.

Beweis. Da R mit der diskreten Topologie versehen ist, ist für jedes Element $r \in R$ ist das System $\{\{r\}\}$ eine Umgebungsbasis. Sei

$$\pi_m : \prod_{\tilde{m}\in M_V} R \to R, \big(r_{\tilde{m}}\big)_{\tilde{m}\in M_V} \mapsto r_m$$

für alle Monome $m \in M_V$ die Projektion von $\prod_{\tilde{m}\in M_V} R$ auf den Koeffizienten von m, dann gilt aufgrund der Konstruktion der Produkttopologie, dass für alle $(r_m)_{m\in M_V} \in \prod_{m\in M_V} R$ das System

$$\begin{aligned}
\mathcal{B}((r_m)_{m\in M_V}) &= \Big\{ \bigcap_{m\in I} \pi_m^{-1}\big(\{r_m\}\big) \,:\, I \subseteq M_V, I \text{ endlich}\Big\} \\
&= \Big\{ \bigcap_{m\in I} \big\{(s_k)_{k\in M_V} \,:\, s_m = r_m\big\} \,:\, I \subseteq M_V, I \text{ endlich}\Big\} \\
&= \Big\{ \big\{(s_k)_{k\in M_V} \in \prod_{m\in M_V} R \,:\, s_m = r_m \text{ für alle } m \in I\big\} \,:\, I \subseteq M_V, I \text{ endlich}\Big\}
\end{aligned}$$

eine Umgebungsbasis von $(r_m)_{m\in M_V}$ ist.

Durch Anwendung der Identifizierungsabbildung ι aus Definition 2.1.5 erhalten wir daher, dass für alle $f \in R[\![V]\!]$ das Mengensystem

$$\mathcal{B}(f) = \big\{\{g \in R[\![V]\!] \,:\, f(m) = g(m) \text{ für alle } m \in I\} \,:\, I \subseteq M_V, I \text{ endlich}\big\}$$

eine Umgebungsbasis von f ist. □

Satz 2.1.7. *Sei $(X, \preceq)$ eine gerichtete Menge und $(f_n)_{n\in X}$ ein Netz mit $f_n \in R[\![V]\!]$ für alle $n \in X$. Das Netz $(f_n)_{n\in X}$ konvergiert genau dann gegen $f \in R[\![V]\!]$, wenn für jedes Monom $m \in M_V$ ein Index N_m existiert, sodass für alle $n \succeq N_m$ gilt, dass $f_n(m) = f(m)$.*

Beweis. Laut Definition des Grenzwertes eines Netzes gilt $\lim f_n = f$ genau dann, wenn für alle $U \in \mathcal{B}(f)$ ein Index $N_U \in X$ existiert, sodass für alle $n \succeq N_U$ gilt, dass $f_n \in U$. Anders formuliert gilt also

$$\lim f_n = f \Leftrightarrow \forall I \subseteq M_V, I \text{ endlich} \;:\; \exists N_I \in X \;\; \forall n \succeq N_I \;\; \forall m \in I : f_n(m) = f(m).$$

Aus der Konvergenz folgt also insbesonders, dass für jedes Monom m ein Index N_m existiert, sodass für alle $n \succeq N_m$ gilt, dass $f_n(m) = f(m)$, indem man die Menge $I := \{m\}$ setzt.

Gelte nun andererseits, dass für jedes Monom $m \in M_V$ ein Index N_m existiert, sodass für alle $n \succeq N_m$ gilt, dass $f_n(m) = f(m)$, und sei $I = \{m_1, \ldots, m_k\}$ eine endliche Teilmenge von

M_V. Es existieren also Indizes $N_{m_1}, \ldots, N_{m_k}$, sodass für alle $j \in \{1, \ldots, k\}$ und alle $n \succeq N_{m_j}$ gilt, dass $f_n(m_j) = f(m_j)$. Weiters existiert in der gerichteten Menge X ein Element N mit $N_{m_1}, \ldots, N_{m_k} \preceq N$, und aus der Transitivität der $\preceq$-Relation folgt daher, dass für alle $n \succeq N$ und alle $m \in I$ gilt, dass $f_n(m) = f(m)$. □

Für die Theorie der kombinatorischen Spezies, die in den folgenden Abschnitten vorgestellt wird, ist noch ein zweiter Konvergenzbegriff wichtig, der für formale Potenzreihen auf endlich vielen Unbestimmten mit dem soeben vorgestellten Begriff übereinstimmt.

Definition 2.1.8. Seien $f, g \in R[\![V]\!]$ zwei formale Potenzreihen und $n \in \mathbb{N}_0$, dann haben f und g *Kontakt der Ordnung* n, in Zeichen $f =_n g$, genau dann, wenn für alle Monome $m \in M_V$ mit $\deg(m) \leq n$ gilt, dass $f(m) = g(m)$.

Satz 2.1.9. *Sei $(f_n)_{n\in\mathbb{N}_0}$ eine Folge von formalen Potenzreihen mit $f_n \in R[\![V]\!]$, wobei V endlich ist, und $f \in R[\![V]\!]$, dann gilt $\lim_{n\to\infty} f_n = f$ genau dann, wenn für alle $k \in \mathbb{N}_0$ ein Index $N \in \mathbb{N}_0$ existiert, sodass für alle $n \geq N$ gilt, dass $f_n =_k f$.*

Beweis. Gelte zuerst, dass $f_n \to f$, und sei $m \in M_V$ ein beliebiges Monom mit $\deg(m) \leq k$, dann existiert aufgrund der Konvergenz ein Index N_m, sodass für alle $n \geq N_m$ gilt, dass $f_n(m) = f(m)$. Da aufgrund der Endlichkeit von V nur endlich viele Monome $m \in M_V$ mit $\deg(m) \leq k$ existieren, können wir $N := \max\{N_m : m \in M_V, \deg(m) \leq k\}$ setzen, und es gilt für alle $n \geq N$, dass $f_n =_k f$.

Gelte nun andererseits, dass für alle $k \in \mathbb{N}_0$ ein Index $N \in \mathbb{N}_0$ existiert, sodass für alle $n \geq N$ gilt, dass $f_n =_k f$, und sei $m \in M_V$ ein beliebiges Monom. Dann existiert insbesonders ein $N_m \in \mathbb{N}_0$, sodass $f_n =_{\deg(m)} f$ für alle $n \geq N_m$, woraus $f_n(m) = f(m)$ für alle $n \geq N_m$ folgt. Da m beliebig war, folgt $\lim_{n\to\infty} f_n = f$. □

Wir wenden uns nun dem Verhalten von Reihen von formalen Potenzreihen zu.

Satz 2.1.10. *Sei $(f_n)_{n\in\mathbb{N}_0}$ eine Folge von formalen Potenzreihen, $f_n \in R[\![V]\!]$. Die Folge der Partialsummen $(s_n)_{n\in\mathbb{N}_0}$, $s_n := \sum_{k=0}^{n} f_n$ konvergiert genau dann, wenn für jedes Monom $m \in M_V$ ein Index K_m existiert, sodass $f_k(m) = 0$ für alle $k > K_m$. Im Fall der Konvergenz ist der Grenzwert die formale Potenzreihe $s : M_V \to R, m \mapsto s_{\min N_m}(m)$, wobei wir definieren, dass $N_m := \{K \in \mathbb{N}_0 : f_k(m) = 0 \quad \text{für alle } k > K\}$. Weiters gilt im Fall der Konvergenz von $(s_n)_{n\in\mathbb{N}_0}$, dass für jede bijektive Abbildung $\sigma : \mathbb{N}_0 \to \mathbb{N}_0$ die Umordnung $(\tilde{s}_n)_{n\in\mathbb{N}_0}$, $\tilde{s}_n := \sum_{k=0}^{n} f_{\sigma(n)}$, gegen den selben Grenzwert konvergiert, dass also $\lim_{n\to\infty} s_n = \lim_{n\to\infty} \tilde{s}_n = s$ gilt.*

Beweis. Sei zuerst $(f_n)_{n\in\mathbb{N}_0}$ konvergent. Aus Satz 2.1.7 folgt, dass für jedes Monom $m \in M_V$ ein Index K_m und $a_m \in R$ existiert, sodass $s_k(m) = a_m$ für alle $k \geq K_m$. Da $f_n(m) = s_n(m) - s_{n-1}(m)$ für alle $n \geq 1$, folgt, dass $f_k(m) = s_k(m) - s_{k-1}(m) = a_m - a_m = 0$ für alle $k > K_m$ gilt. Gilt andererseits, dass für jedes Monom $m \in M_V$ ein Index K_m existiert, sodass $f_k(m) = 0$ für alle $k > K_m$, so können wir die formale Potenzreihe $s \in R[\![V]\!]$ definieren, die jedes Monom m auf $s(m) := s_{\min N_m}(m)$ abbildet, und es gilt $s_k(m) = s(m)$ für alle $k > K_m$. Die Partialsummenfolge $(s_n)_{n\in\mathbb{N}_0}$ konvergiert also gegen s.

Sei nun $(s_n)_{n\in\mathbb{N}_0}$ konvergent mit Grenzwert s und $\sigma : \mathbb{N}_0 \to \mathbb{N}_0$ eine bijektive Abbildung. Betrachten wir ein beliebiges Monom $m \in \mathbb{N}_0$, dann existiert ein Index K_m, sodass für alle $k > K_m$ gilt, dass $f_k(m) = 0$. Es gibt also nur endlich viele Indizes k_i, sodass $f_{k_i}(m) \neq 0$ ist. Daher existieren auch nur endlich viele Indizes $\tilde{k}_i$, sodass $f_{\sigma(\tilde{k}_i)} \neq 0$ ist. Also gibt es einen Index $\tilde{K}_m$, sodass $f_{\sigma(k)} = 0$ für alle $k > \tilde{K}_m$ gilt, und die Partialsummenfolge $(\tilde{s}_n)_{n\in\mathbb{N}_0}$ konvergiert ebenfalls, und zwar gegen den Grenzwert $\tilde{s}$ mit $\tilde{s}(m) := \tilde{s}_{\min \tilde{N}_m}(m)$, wobei $\tilde{N} := \left\{K \in \mathbb{N}_0 : f_{\sigma(k)}(m) = 0 \text{ für alle } k > K\right\}$. Um zu zeigen, dass der Grenzwert $s = \tilde{s}$ ist, betrachten wird die Menge $I_m := \{k \in \mathbb{N}_0 : f_k(m) \neq 0\}$ aller Indizes k, für die $f_k(m) \neq 0$ ist, die also zur Summe etwas beitragen. Wir sehen leicht, dass $\sigma^{-1}(I_m) = \left\{k \in \mathbb{N}_0 : f_{\sigma(k)} \neq 0\right\}$ die Menge aller Indizes ist, die zur umgeordneten Summe etwas beitragen. Es gilt daher, dass

$$s(m) = \sum_{k=0}^{\min N_k} f_k(m) = \sum_{k\in I_m} f_k(m) = \sum_{k\in\sigma^{-1}(I_m)} f_{\sigma(k)}(m) = \sum_{k=0}^{\min \tilde{N}_k} f_{\sigma(k)}(m) = \tilde{s}(m).$$

□

Bemerkung. Da im Fall der Konvergenz die Summationsreihenfolge also immer unbedeutend ist, werden wir in Folge die Summe einer endlichen oder abzählbar unendlichen Menge bzw Familie von formalen Potenzreihen als Grenzwert der Partialsummenfolge bezüglich einer Aufzählung auffassen, sofern dieser existiert. Sei also $A \subseteq R[\![V]\!]$, dann definieren wir im Fall, dass A endlich ist, und $f_1, \ldots, f_n$ eine Aufzählung ist, die Summe als

$$\sum_{f \in A} f := \sum_{i=1}^{n} f_i.$$

Falls A abzählbar unendlich ist, existiert eine Folge $\big(f_n\big)_{n \in \mathbb{N}_0}$ mit $A = \{f_n : n \in \mathbb{N}_0\}$, und wir definieren

$$\sum_{f \in A} f := \lim_{n \to \infty} \sum_{k=0}^{n} f_k,$$

falls dieser Grenzwert existiert. Der Fall der Familie von formalen Potenzreihen wird analog behandelt.

Insbesonders rechtfertigt diese Definition die in der Literatur gewohnte Darstellung einer formalen Potenzreihe $f \in R[\![V]\!]$ als

$$\sum_{z_{i_1}^{n_1} \cdots z_{i_k}^{n_k} \in M_V} \big[z_{i_1}^{n_1} \cdots z_{i_k}^{n_k}\big] f \;\; z_{i_1}^{n_1} \cdots z_{i_k}^{n_k}.$$

Nun betrachten wir noch die sogenannte *Einsetzungs-* oder *Substitutionsoperation*. Sie ist das Analogon zur Funktionskomposition aus der Analysis.

Definition 2.1.11. Seien V und W endliche oder abzählbar unendliche Mengen, $m := z_{i_1}^{n_1} \cdots z_{i_k}^{n_k} \in R[\![V]\!]$ ein Monom und $\big(g_v\big)_{v \in V}$ eine Familie von formalen Potenzreihen mit $g_v \in R[\![W]\!]$ für alle $v \in V$, so ist die *Einsetzungs-* beziehungsweise *Substitutionsoperation*, in Zeichen $m \circ \big(g_v\big)_{v \in V}$, als

$$m \circ \big(g_v\big)_{v \in V} := \prod_{\substack{v \in V \\ m(v) \neq 0}} g_v^{m(v)} = \big(g_{z_{i_1}}\big)^{n_1} \cdots \big(g_{z_{i_k}}\big)^{n_k}$$

definiert, wobei $m \circ \big(g_v\big)_{v \in V} \in R[\![W]\!]$ gilt. Sei weiters $f \in R[\![V]\!]$ eine beliebige formale Potenzreihe und M_V die Menge der Monome auf V, dann ist die Einsetzungsoperation $f \circ \big(g_v\big)_{v \in V}$ als

$$f \circ \big(g_v\big)_{v \in V} := \sum_{m \in M_V} f(m) \;\; m \circ \big(g_v\big)_{v \in V}$$

definiert, falls diese Summe existiert. Falls die verwendeten Variablen klar sind, schreiben wir die Operation auch gerne in der Form

$$f \circ \big(g_v\big)_{v \in V} = f\big(g_{z_1}(x_1, x_2, \ldots), g_{z_2}(x_1, x_2, \ldots), \ldots\big).$$

2.2 Spezies von Strukturen

Nun wenden wir uns den sogenannten *kombinatorischen Spezies* zu. Diese stellen eine Möglichkeit dar, den intuitiv klaren Strukturbegriff, mit dem in der Kombinatorik gearbeitet wird, mathematisch exakt zu definieren. Eine kombinatorische Spezies besteht aus einer Abbildungsvorschrift, die aus einer beliebigen gegebenen endliche Menge von Grundbausteinene Strukturen erzeugt. Weiters ordnet die Spezies jeder Bijektion zwischen zwei gleichmächtigen Grundmengen eine Bijektion zwischen den erzeugten Strukturen zu. Diese informelle Erklärung führt uns zu der folgenden Definition.

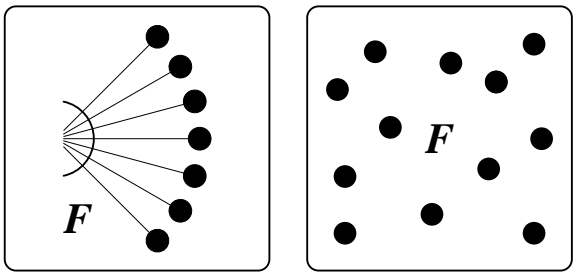

Abbildung 1: Symbolische Darstellungen einer Spezies von Strukturen

Definition 2.2.1. Eine *Spezies von Strukturen* ist eine Regel $\mathcal{F}$, die für jede endliche Menge U eine endliche Menge $\mathcal{F}[U]$ erzeugt, und jeder Bijektion $\sigma : U \to V$ eine Funktion $\mathcal{F}[\sigma] : \mathcal{F}[U] \to F[V]$ zuordnet. Die Funktionen $\mathcal{F}[\sigma]$ müssen dabei folgende Eigenschaften besitzen:

- Für alle endlichen Mengen U, V, W und alle Bijektionen $\sigma : U \to V$ und $\tau : V \to W$ gilt
$$\mathcal{F}[\tau \circ \sigma] = \mathcal{F}[\tau] \circ \mathcal{F}[\sigma]. \tag{1}$$
- Für alle endlichen Mengen U gilt
$$\mathcal{F}[\mathrm{id}_U] = \mathrm{id}_{\mathcal{F}[U]}, \text{ wobei id die Identität bezeichnet.} \tag{2}$$

Ein Element $s \in \mathcal{F}[U]$ wird als *$\mathcal{F}$-Struktur auf U* oder auch als *Struktur der Spezies $\mathcal{F}$ auf U* bezeichnet. Die Funktion $\mathcal{F}[\sigma]$ heißt *Transport der $\mathcal{F}$-Strukturen entlang σ*. Wenn wir die Funktion $\mathcal{F}[\sigma]$ für eine Struktur s auswerten, schreiben wir gerne auch $\sigma \cdot s$ anstatt $\mathcal{F}[\sigma](s)$.

Bemerkung. Aufgrund der angeführten Eigenschaften sehen wir, dass jede Transport-Funktion $\mathcal{F}[\sigma]$ eine Bijektion sein muss, weil für beliebige endliche Mengen U und V und jede beliebige Bijektion $\sigma : U \to V$

$$\mathrm{id}_{\mathcal{F}[V]} = \mathcal{F}[\mathrm{id}_V] = \mathcal{F}[\sigma \circ \sigma^{-1}] = \mathcal{F}[\sigma] \circ \mathcal{F}[\sigma^{-1}], \text{ und}$$
$$\mathrm{id}_{\mathcal{F}[U]} = \mathcal{F}[\mathrm{id}_U] = \mathcal{F}[\sigma^{-1} \circ \sigma] = \mathcal{F}[\sigma^{-1}] \circ \mathcal{F}[\sigma] \text{ gilt.}$$

Wenn U und V zwei gleich mächtige endliche Mengen sind und daher eine Bijektion $\tau : U \to V$ existiert, sehen wir also weiters, dass $\mathcal{F}[U]$ und $\mathcal{F}[V]$ gleich mächtig sein müssen, weil $\mathcal{F}[\tau]$ eine Bijektion zwischen $\mathcal{F}[U]$ und $\mathcal{F}[V]$ ist. Die Anzahl der Strukturen, die eine Spezies erzeugt, ist also nur von der Mächtigkeit der zugrundeliegenden Menge abhängig, nicht jedoch von speziellen Eigenschaften der Elemente der Menge.

Ein Vorteil dieser Definition ist, dass die Regel, wie Strukturen erzeugt werden und wie der Transport der Strukturen abläuft, auf verschiedenste Art und Weise angegeben werden kann. Das kann zum Beispiel durch eine Axiomenmenge, die erfüllt werden muss, durch Angabe eines Algorithmus, der die Strukturen erzeugt, oder durch explizite Angabe der Menge der Strukturen erfolgen. Eine weitere sehr wichtige Möglichkeit ist die Definition einer Spezies mittels Anwendung von Operationen auf bereits definierte Spezies.
Wir betrachten nun einige grundlegende Spezies, die wir im weiteren Verlauf verwenden werden.

Definition 2.2.2. Seien im U und V beliebige endliche Mengen, $\sigma : U \to V$ eine beliebige Bijektion, und $f_\emptyset$ die leere Funktion $f_\emptyset : \emptyset \to \emptyset$, dann gilt:

- Die *Spezies der Singletons*, auch als $\mathcal{Z}$ bezeichnet, bildet aus U die Strukturen
$$\mathcal{Z}[U] := \begin{cases} \{U\} & \text{falls } |U| = 1, \\ \emptyset & \text{sonst.} \end{cases}$$
Der Transport der Strukturen ist für den Fall, dass $|U| = |V| = 1$ als $\mathcal{Z}[\sigma](U) = V$, ansonsten als $\mathcal{Z}[\sigma][s] = \emptyset$ definiert.

- Die *Spezies der Charakteristik der leeren Mengen*, in Zeichen $\mathbf{1}$, erzeugt aus U die Strukturen

$$\mathbf{1}[U] := \begin{cases} \{\emptyset\} & \text{falls } |U| = 0, \\ \emptyset & \text{sonst.} \end{cases}$$

  Für σ gilt $\mathbf{1}[\sigma](\emptyset) = \emptyset$, falls $|U| = 0$, ansonsten $\mathbf{1}[\sigma] = f_\emptyset$.

- Die *leere Spezies* $\mathbf{0}$ erzeugt aus U die Strukturen $\mathbf{0}[U] := \emptyset$, also gar keine. Der Transport der Strukturen ist daher immer die leere Funktion.

- Die *Spezies der Mengen*, auch als SET bezeichnet, bildet aus U die Strukturen $\textsc{Set}[U] := \{U\}$. Für σ gilt, dass $\textsc{Set}[\sigma](U) = V$.

- Für die *Spezies der Folgen* SEQ gilt

$$\textsc{Seq}[U] := \begin{cases} \{\emptyset\} & \text{für } |U| = 0, \\ \{(\pi(1), \pi(2), \ldots, \pi(|U|)) \,:\, \pi : \{1, \ldots, |U|\} \to U \text{ bijektiv}\} & \text{sonst.} \end{cases}$$

  Für den Transport der Strukturen gilt für den Fall, dass $|U| > 0$,

$$\textsc{Seq}[\sigma]\left((\pi(1), \ldots, \pi(|U|))\right) = (\sigma \circ \pi(1), \ldots, \sigma \circ \pi(|V|))\,,$$

  ansonsten $\textsc{Seq}[\sigma](\emptyset) = \emptyset$.

- Die *Spezies der Permutationen* $\mathcal{P}$ erzeugt aus der Menge U die Strukturen

$$\mathcal{P}[U] := \{\pi : U \to U \,:\, \pi \text{ bijektiv}\}\,.$$

  Der Transport der Strukturen ist als

$$\mathcal{P}[\sigma](\pi) := \sigma \circ \pi \circ \sigma^{-1} \text{ für alle } \pi \in \mathcal{P}[U]$$

  definiert. Im Folgenden bezeichnen wir die Menge der Permutationen der Menge $\{1, \ldots, n\}$ als $\mathcal{P}_n := \mathcal{P}[\{1, \ldots, n\}]$.

- Die *Spezies der Zyklen*, auch als CYC bezeichnet, ordnet der Menge U die Strukturen

$$\textsc{Cyc}[U] := \begin{cases} \emptyset & \text{für } U = \emptyset, \\ \{\pi \in \mathcal{P}[U] \,:\, \pi \text{ besteht aus genau einem Zyklus}\} & \text{sonst,} \end{cases}$$

  zu. Der Transport der Strukturen ist als

$$\textsc{Cyc}[\sigma] := \begin{cases} f_\emptyset & \text{für } U = \emptyset, \\ \mathcal{P}[\sigma]\big|_{\textsc{Cyc}[U]} & \text{sonst,} \end{cases}$$

  definiert.

- Die *Spezies der Partitionen* PAR ordnet der Menge U die Strukturen

$$\textsc{Par}[U] := \begin{cases} \{\emptyset\} \text{ für } U = \emptyset, \\ \left\{\pi \subseteq \mathbf{2}^U \,:\, \emptyset \notin \pi, A \cap B = \emptyset \text{ für alle } A, B \in \pi, \text{ und } \bigcup_{A \in \pi} A = U\right\} \text{ sonst,} \end{cases}$$

  zu. Der Transport der Strukturen ist als

$$\textsc{Par}[\sigma]\left(\{P_1, \ldots, P_k\}\right) := \{\sigma(P_1), \ldots, \sigma(P_k)\}$$

  definiert.

Nun wenden wir uns nun dem Problem der unmarkierten Strukturen zu. Oft sind nicht die konkreten Strukturen, sondern Klassen von Strukturen interessant, die informell durch Weglassen der Beschriftung entstehen. Eine derartige Klasse wird dann *unmarkierte Struktur* bzw. *Isomorphietyp* genannt. Die formale Konstruktion dieser Klassen wird im Folgenden beschrieben.

Definition 2.2.3. Seien $s_1 \in \mathcal{F}[U]$ und $s_2 \in \mathcal{F}[V]$, dann heißt eine Bijektion $\sigma : U \to V$ *Isomorphismus zwischen s_1 und s_2*, genau dann, wenn $\sigma \cdot s_1 = s_2$. Die Strukturen s_1 und s_2 haben genau in diesem Fall den selben *Isomorphietyp*. Für eine Struktur s nennen wir weiters einen Isomorphismus zwischen s und s einen *Automorphismus*.

Definition 2.2.4. Sei U eine endliche Menge und $\mathcal{F}$ eine Spezies, dann heißt die Relation $\sim_U$ auf $\mathcal{F}[U]$ *Isomorphietyprelation*. Für zwei Strukturen s_1 und s_2 gilt

$$s_1 \sim_U s_2 :\Leftrightarrow \text{Es existiert eine Bijektion } \sigma : U \to U \text{ mit } \mathcal{F}[\sigma](s_1) = s_2.$$

Wenn die Grundmenge U aus dem Kontext klar ist, wird die Relation auch einfach nur als $\sim$ geschrieben.

Satz 2.2.5. *Sei U eine beliebige endliche Menge und $\mathcal{F}$ eine Spezies, dann ist die Relation $\sim$ eine Äquivalenzrelation. Die Äquivalenzklassen der Menge $\mathcal{F}[U]/\sim$ werden als* Isomorphietypen *bezeichnet. Wenn V eine weitere endliche Menge und $\sigma : U \to V$ eine Bijektion ist, dann gilt*

$$s \sim_U t \Leftrightarrow \sigma \cdot s \sim_V \sigma \cdot t.$$

Beweis. Zuerst zeigen wir, dass $\sim$ eine Äquivalenzrelation ist.

- Die Reflexivität folgt daraus, dass für eine beliebige Struktur $s \in \mathcal{F}[U]$ gilt, dass $\mathrm{id} \cdot s = s$. Es folgt also $s \sim s$ für alle $s \in \mathcal{F}[U]$.
- Wenn nun s und t zwei Strukturen aus $\mathcal{F}[U]$ mit $s \sim t$ sind, dann existiert eine Bijektion $\pi : U \to U$ mit $\pi \cdot s = t$. Aufgrund der Definition des Transports der Strukturen folgt also weiter, dass $\pi^{-1} \cdot (\pi \cdot s) = (\pi^{-1} \circ \pi) \cdot s = \mathrm{id} \cdot s = s = \pi^{-1} \cdot t$ ist. Es gibt also eine Permutation $\tau : U \to U := \pi^{-1}$ mit $\tau \cdot t = s$, woraus $t \sim s$ und somit die Symmetrie folgt.
- Um die Transitivität zu zeigen, betrachten wir drei beliebige Strukturen s, t und u aus $\mathcal{F}[U]$, für die $s \sim t$ und $t \sim u$ gilt. Es existieren also Bijektionen $\pi : U \to U$ und $\tau : U \to U$ mit $\pi \cdot s = t$ und $\tau \cdot t = u$. Daher folgt $\tau \cdot t = \tau \cdot (\pi \cdot s) = (\tau \circ \pi) \cdot s = u$. Für die Bijektion $\sigma := \tau \circ \pi$ erhalten wir also $\sigma \cdot s = u$ und somit $s \sim u$.

Nun wenden wir uns der zweiten Aussage des Satzes zu. Seien U und V endliche Mengen und $\sigma : U \to V$ bijektiv. Weiters seien s und t zwei Strukturen aus $\mathcal{F}[U]$, für die $s \sim_U t$ gilt. Es existiert also eine Bijektion $\pi : U \to U$ mit $F[\pi](s) = t$. Nun folgt

$$\begin{aligned}\mathcal{F}[\pi](s) = t \Rightarrow \mathcal{F}[\sigma]\left(\mathcal{F}[\pi](s)\right) = \mathcal{F}[\sigma](t) \Rightarrow \mathcal{F}[\sigma \circ \pi](s) = \mathcal{F}[\sigma](t)\\ \Rightarrow \mathcal{F}[\sigma \circ \pi \circ \sigma^{-1} \circ \sigma](s) = \mathcal{F}[\sigma](t) \Rightarrow \mathcal{F}[\sigma \circ \pi \circ \sigma^{-1}]\left(\mathcal{F}[\sigma](s)\right) = \mathcal{F}[\sigma](t).\end{aligned}$$

Mit $\tau := \sigma \circ \pi \circ \sigma^{-1}$ sehen wir also, dass $\tau \cdot (\sigma \cdot s) = \sigma \cdot t$ und somit $\sigma \cdot s \sim_V \sigma \cdot t$. Die andere Richtung der Äquivalenz folgt analog. □

Definition 2.2.6. Sei $\mathcal{F}$ eine Spezies und U eine endliche Menge, dann bezeichnet $T(\mathcal{F}, U)$ die Menge der Isomorphietypen $\mathcal{F}[U]/\sim_U$. Ist $U = [n]$, dann schreiben wir abkürzend $T(\mathcal{F}, n)$ an Stelle von $T(\mathcal{F}, [n])$.

Zwei Strukturen eines Isomorphietyps gehen also durch *Neuverteilung der Markierungen* ineinander über. Wir können das so interpretieren, dass die Isomorphietypen Strukturen zusammenfassen, bei denen es auf die Markierung der sie erzeugenden Bauteile nicht ankommt. Weiters erkennen wir aus der zweiten Aussage des Satzes, dass der Transport der Strukturen mit den Isomorphietypen kompatibel ist. Insbesonders ist die Anzahl der Isomorphietypen nur von der Kardinalität der Grundmenge, nicht von der Menge selber abhängig.

Nun betrachten wir noch verschiedene Begriffe der Gleichheit, die im Kontext der Speziestheorie auftreten. Der erste und restriktivste Begriff ist die Identität.

Definition 2.2.7. Zwei Spezies $\mathcal{F}$ und $\mathcal{G}$ heißen *gleich* beziehungsweise *ident*, wenn sie die gleichen Strukturen erzeugen, und wenn der Transport der Strukturen übereinstimmt, wenn also gilt:

$$\mathcal{F}[U] = \mathcal{G}[U] \text{ für alle endlichen Mengen } U \text{ und} \tag{3}$$

$$\mathcal{F}[\sigma] = \mathcal{G}[\sigma] \text{ für alle endlichen Mengen } U \text{ und } V \text{ und alle Bijektionen } \sigma : U \to V. \tag{4}$$

Wenn die Spezies $\mathcal{F}$ und $\mathcal{G}$ gleich sind, dann schreiben wir auch $\mathcal{F} = \mathcal{G}$.

Aufgrund der Restriktivität wird dieser Gleichheitsbegriff im Folgenden von wenig Bedeutung sein. Der gröbste verwendete Gleichheitsbegriff bezieht sich auf die Anzahl der Strukturen, die die Spezies erzeugen.

Definition 2.2.8. Die Spezies $\mathcal{F}$ und $\mathcal{G}$ werden genau dann als *äquipotent* bezeichnet, wenn für alle endlichen Mengen U gilt, dass $|\mathcal{F}[U]| = |\mathcal{G}[U]|$. Wir schreiben dann $\mathcal{F} \equiv \mathcal{G}$.

Dieser Gleichheitsbegriff hat allerdings den Nachteil, dass zwar die Anzahl der erzeugten Strukturen gleich ist, diese Gleichheit aber bei den Isomorphietypen endet.

So sind zwar die Spezies der Permutationen und die Spezies der Folgen equipotent, wie man leicht sieht, wenn mann jede Folge mit der einzeiligen Darstellung der entsprechenden Permutation identifiziert, allerdings ist die Anzahl der Isomorphietypen im Allgemeinen nicht gleich.

Sei U eine beliebige endliche Menge mit $|U| > 1$. Für je zwei beliebige Folgen $s_1, s_2 \in \textsc{Seq}[U]$ gilt mit $s_1 = (\pi_1(1), \ldots, \pi_1(|U|))$ für eine Bijektion $\pi_1 : \{1, \ldots, |U|\} \to U$ und $s_2 = (\pi_2(1), \ldots, \pi_2(|U|))$ für eine Bijektion $\pi_2 : \{1, \ldots, |U|\} \to U$, dass $\textsc{Seq}[\sigma](s_1) = s_2$, wenn man $\sigma := \pi_2 \circ \pi_1^{-1}$ setzt. Es sind also alle Folgen aus $\textsc{Seq}[U]$ zueinander isomorph, und es gibt daher jeweils nur einen Isomorphietyp.

Betrachten wir auf der anderen Seite eine beliebige Permutation $\sigma : U \to U$, dann gilt für alle Bijektionen $\phi : U \to U$, dass $\phi \circ \sigma \circ \phi^{-1} = \mathrm{id}_U$ genau dann, wenn $\sigma = \phi^{-1} \circ \phi = \mathrm{id}_U$. Es gibt daher mindestens zwei Isomorphietypen, denn die Identität ist nur isomorph zu sich selbst.

Dieser Sachverhalt wird in den Abbildungen 2.2 und 2.2 veranschaulicht.

Der für die meisten Fälle relevante Gleichheitsbegriff liegt zwischen diesen beiden Extremen.

Definition 2.2.9. Seien $\mathcal{F}$ und $\mathcal{G}$ zwei beliebige Spezies. Ein *Isomorphismus* von $\mathcal{F}$ nach $\mathcal{G}$ ist eine durch alle endlichen Mengen indizierte Familie von Bijektionen $\alpha_U : \mathcal{F}[U] \to \mathcal{G}[U]$, sodass für jede Bijektion $\sigma : U \to V$ zwischen zwei endlichen Mengen U und V das folgende Diagramm kommutiert:

$$\begin{array}{ccc} \mathcal{F}[U] & \xrightarrow{\alpha_U} & \mathcal{G}[U] \\ {\scriptstyle \mathcal{F}[\sigma]}\downarrow & & \downarrow{\scriptstyle \mathcal{G}[\sigma]} \\ \mathcal{F}[V] & \xrightarrow{\alpha_V} & \mathcal{G}[V] \end{array} \tag{5}$$

Es muss also für jede $\mathcal{F}$-Struktur s gelten, dass $\sigma \cdot \alpha_U(s) = \alpha_V(\sigma \cdot s)$. Die Spezies $\mathcal{F}$ und $\mathcal{G}$ werden genau dann *isomorph* bzw. *kombinatorisch gleich* genannt, wenn ein Isomorphismus von $\mathcal{F}$ nach $\mathcal{G}$ existiert. Wir schreiben dann auch $\mathcal{F} \simeq \mathcal{G}$.

Da zwei kombinatorische Spezies, die isomorph sind, im wesentlichen die gleichen kombinatorischen Eigenschaften besitzen, werden wir in Zukunft zwei Spezies $\mathcal{F}$ und $\mathcal{G}$ genau dann als gleich betrachten, wenn sie isomorph sind. Wir schreiben dann $\mathcal{F} = \mathcal{G}$ anstatt $\mathcal{F} \simeq \mathcal{G}$.

Der letzte Gleichheitsbegriff, den wir betrachten, ist ein topologischer, der bei Konvergenzbetrachtungen von Folgen von Spezies verwendet wird.

Definition 2.2.10. Seien $\mathcal{F}$ und $\mathcal{G}$ zwei kombinatorische Spezies und $n \in \mathbb{N}_0$, dann haben $\mathcal{F}$ und $\mathcal{G}$ *Kontakt der Ordnung* n, in Zeichen $\mathcal{F} =_n \mathcal{G}$, genau dann, wenn $\mathcal{F}_{\leq n} = \mathcal{G}_{\leq n}$ gilt. Für eine Spezies $\mathcal{F}$ bezeichnet die Spezies $\mathcal{F}_{\leq n}$ die *Einschränkung der Spezies* $\mathcal{F}$ *auf Mengen der Kardinalität* n, die auf einer endlichen Menge U die Strukturen

$$\mathcal{F}_{\leq n}[U] = \begin{cases} \mathcal{F}[U] & \text{für } |U| \leq n, \\ \emptyset & \text{sonst,} \end{cases}$$

bildet, und deren Transport der Strukturen für eine weitere endliche Menge V und eine beliebige Bijektion $\sigma : U \to V$ als

$$\mathcal{F}_{\leq n}[\sigma] = \begin{cases} \mathcal{F}[\sigma] & \text{für } |U| \leq n, \\ f_\emptyset & \text{sonst,} \end{cases}$$

definiert ist.

Eine Folge von Spezies $(\mathcal{F}_n)_{n\in\mathbb{N}}$ konvergiert gegen eine Spezies $\mathcal{F}$, wenn die Folge der Einschränkungen der $\mathcal{F}_n$ auf Mengen der Kardinalität N schließlich konstant im Sinne der kombinatorischen Gleichheit ist. Das heißt, dass auf einer vorgegebenen endlichen Menge ab einem gewissen Folgenindex keine zusätzlichen Strukturen im Vergleich zu den vorherigen Folgenindizes hinzukommen. Dieser Konvergenzbegriff ist auch mit den im nächsten Abschnitt eingeführten erzeugenden Funktionen kompatibel.

Definition 2.2.11. Eine Folge von kombinatorischen Spezies $(\mathcal{F}_n)_{n\in\mathbb{N}}$ konvergiert genau dann gegen die Spezies $\mathcal{F}$, wenn für alle $N \in \mathbb{N}$ ein $K \in \mathbb{N}$ existiert, sodass für alle $n \geq K$ gilt, dass $\mathcal{F}_n =_N \mathcal{F}$. Wir schreiben dann

$$\lim_{n\to\infty} \mathcal{F}_n = \mathcal{F}. \tag{6}$$

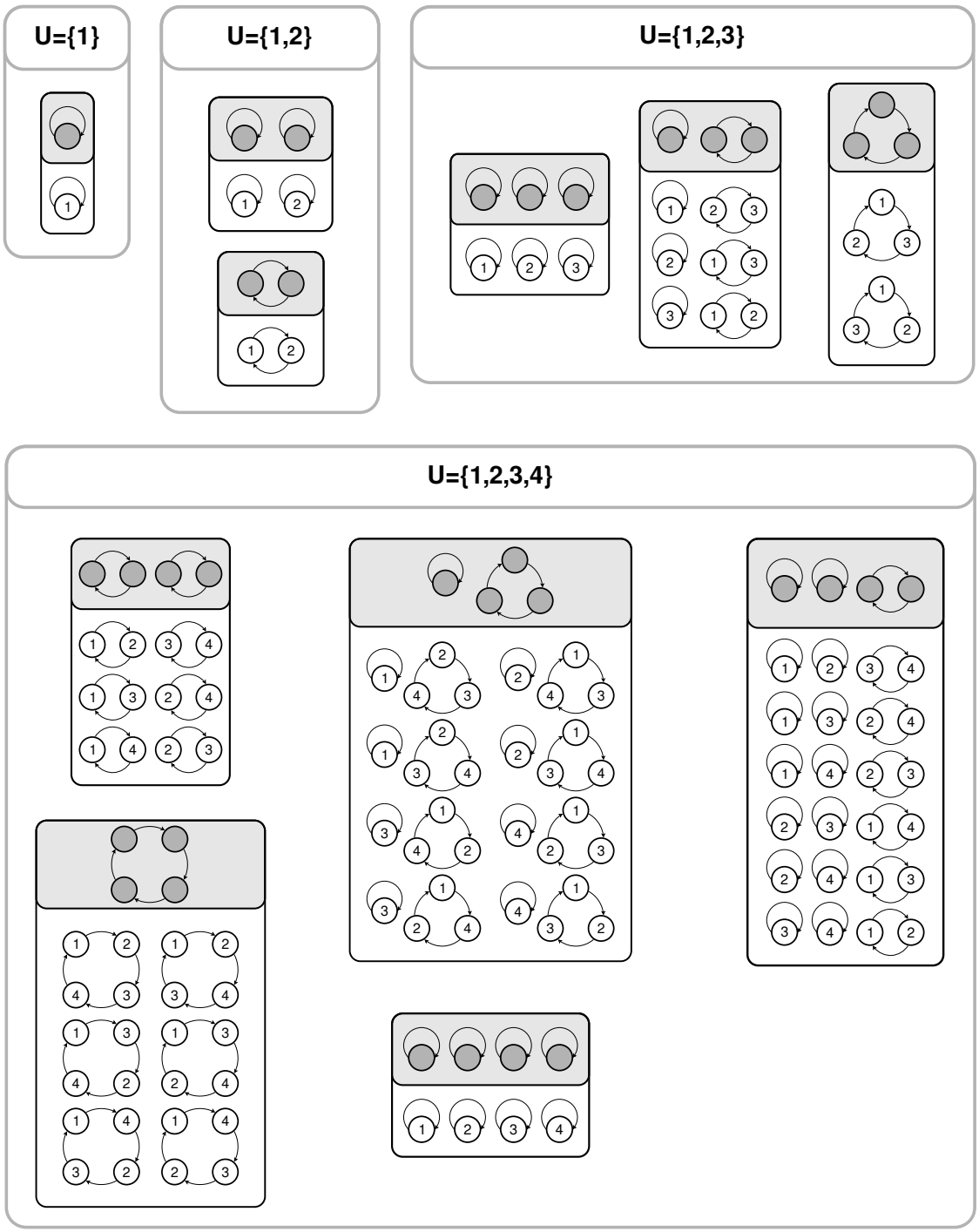

Abbildung 2: Strukturen der Spezies der Permutationen $\mathcal{P}[U]$ mit Isomorphietypen

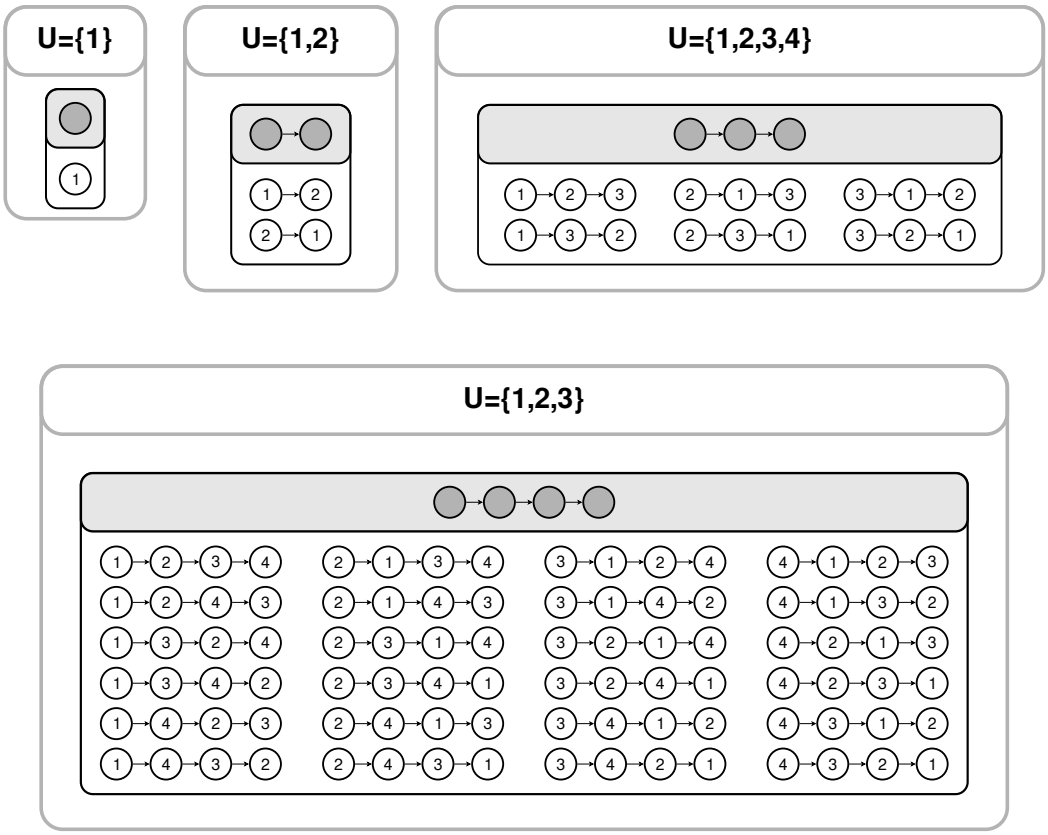

Abbildung 3: Strukturen der Spezies der Folgen $\textsc{Seq}[U]$ mit Isomorphietypen

2.3 Erzeugende Funktionen

Einer kombinatorischen Spezies können formale Potenzreihen zugeordnet werden, die diverse Informationen über die Spezies repräsentieren. Die erste formale Potenzreihe, die wir betrachten, ist die *exponentiell erzeugende Funktion*, die die Anzahl der $\mathcal{F}$-Strukturen repräsentiert.

Definition 2.3.1. Sei $\mathcal{F}$ eine kombinatorische Spezies, dann ist die *exponentiell erzeugende Funktion von* $\mathcal{F}$, auch als $F(z)$ bezeichnet, die formale Potenzreihe

$$F(z) := \sum_{n\geq 0} f_n \frac{z^n}{n!}, \tag{7}$$

wobei $f_n = |\mathcal{F}[n]|$.

Wenn wir nun eine beliebige endliche Menge U mit Kardinalität $n \geq 0$ betrachten, dann gilt $|\mathcal{F}[U]| = n! \cdot [z^n]F(z)$, da aufgrund der gleichen Mächtigkeit von U und $[n]$ eine Bijektion $\sigma : U \to [n]$ und daher auch eine Bijektion $\mathcal{F}[\sigma] : \mathcal{F}[U] \to \mathcal{F}[n]$ existiert. Also gilt $|\mathcal{F}[U]| = |\mathcal{F}[n]|$.

Die nächste formale Potenzreihe, die wir betrachten, ist die *gewöhnliche erzeugende Funktion*, die die Anzahl der Isomorphietypen, also der unmarkierten Strukturen zählt.

Definition 2.3.2. Sei $\mathcal{F}$ eine kombinatorische Spezies, dann ist die *gewöhnliche erzeugende Funktion von* $\mathcal{F}$ als die formale Potenzreihe

$$\widetilde{F}(z) := \sum_{n\geq 0} \tilde{f}_n z^n \tag{8}$$

definiert, wobei $\tilde{f}_n = |T(\mathcal{F}, n)|$. Sie wird auch als *typerzeugende Funktion* bezeichnet.

Aufgrund von Satz 2.2.5 gilt für eine beliebige endliche Menge U mit Mächtigkeit n, dass $|T(\mathcal{F}, U)| = |T(\mathcal{F}, n)|$, also auch $|T(\mathcal{F}, U)| = [z^n]\widetilde{F}(z)$.

Die dritte formale Potenzreihe, die wir betrachten, ist die sogenannte *Zyklenindexreihe*. Diese beinhaltet hinreichend Information, um die bereits definierten erzeugenden Funktionen davon abzuleiten, und wird sich für manche Operationen, die wir im nächsten Kapitel einführen werden, als nützlich erweisen. Zur Definition der Zyklenindexreihe benötigen wir allerdings noch einige Begriffe.

Definition 2.3.3. Sei U eine endliche Menge und $\sigma : U \to U$ eine Permutation. Der *Zyklentyp der Permutation* σ ist die Folge $(\sigma_1, \sigma_2, \sigma_3, \ldots)$, wobei σ_k für $k \geq 1$ die Anzahl der Zyklen der Länge k in der Zerlegung von σ in disjunkte Zyklen angibt.

Bemerkung. Wir sehen, dass σ_1 gleich der Anzahl der Fixpunkte der Permutation σ ist. Außerdem kann eine Permutation auf einer Menge U mit $|U| = n$ keine Zyklen mit einer Länge größer als n besitzen. Daher können wir den Zyklentyp einer derartigen Permutation als n-Tupel $(\sigma_1, \sigma_2, \ldots, \sigma_n)$ schreiben.

Definition 2.3.4. Sei U eine endliche Menge und $\sigma : U \to U$ eine Permutation, dann bezeichnen wir die Menge der Fixpunkte von σ mit

$$\operatorname{Fix} \sigma := \{u \in U \,:\, \sigma(u) = u\}, \tag{9}$$

die Anzahl der Fixpunkte von σ mit

$$\operatorname{fix} \sigma := |\operatorname{Fix} \sigma|. \tag{10}$$

Definition 2.3.5. Sei $\mathcal{F}$ eine kombinatorische Spezies, dann wird die formale Potenzreihe in den unendlich vielen Veränderlichen $z_1, z_2, \ldots$

$$Z_F(z_1, z_2, \ldots) := \sum_{n\geq 0} \frac{1}{n!} \sum_{\sigma\in S_n} \operatorname{fix} \mathcal{F}[\sigma] \cdot z_1^{\sigma_1} z_2^{\sigma_2} \cdots z_n^{\sigma_n}$$

als *Zyklenindexreihe* der Spezies $\mathcal{F}$ bezeichnet. Bei S_n handelt es sich um die symmetrische Gruppe auf der Menge $[n]$ und bei $\sigma_1, \ldots, \sigma_n$ um die Komponenten des Zyklentyps der Permutation σ.

Die Einschränkung auf die Mengen $[n]$ stellt keine Beschränkung der Allgemeinheit dar, wie wir in den folgenden Absätzen zeigen werden. Dazu untersuchen wir zuerst, wie sich die Zyklenstruktur von Permutationen unter Abbildungen verhält.

Lemma 2.3.6. *Seien A und B endliche Mengen mit $|A| = |B| = n$, $\sigma \in S_A$ und $\tau : A \to B$ eine beliebige Bijektion, dann hat die Abbildung $\widetilde{\sigma} := \tau \circ \sigma \circ \tau^{-1} \in S_B$ den selben Zyklenindex wie σ. Seien weiters $\alpha \in S_A$ und $\beta \in S_B$ zwei beliebige Permutationen der entsprechenden Mengen mit gleichem Zyklenindex, dann existiert eine Bijektion $\tau : A \to B$, sodass $\beta = \tau \circ \alpha \circ \tau^{-1}$.*

Beweis. Wir betrachten zuerst den Fall, dass $n > 1$ ist. Für alle $a, b \in A$ gilt, dass sie genau dann im selben Zyklus von σ liegen, wenn $n \in \mathbb{Z}$ existiert, sodass $\sigma^n(a) = b$. Diese Relation schreiben wir als $a \sim_\sigma b$. Es handelt sich um eine Äquivalenzrelation, deren Äquivalenzklassen die Zyklen sind. Nun gilt für alle $a, b \in A$ und für alle $n \in \mathbb{Z}$:

$$\begin{aligned}\sigma^n(a) = b &\Leftrightarrow \tau(\sigma^n(a)) = \tau(b) \Leftrightarrow \tau(\sigma^n(\tau^{-1}(\tau(a)))) = \tau(b)\\ &\Leftrightarrow (\tau \circ \sigma^n \circ \tau^{-1})(\tau(a)) = \tau(b) \Leftrightarrow (\tau \circ \sigma \circ \tau^{-1})^n(\tau(a)) = \tau(b)\\ &\Leftrightarrow \widetilde{\sigma}^n(\tau(a)) = \tau(b).\end{aligned}$$

Also gilt auch für alle $a, b \in A$, dass

$$a \sim_\sigma b \Leftrightarrow \tau(a) \sim_{\widetilde{\sigma}} \tau(b).$$

Die Bijektion τ bildet daher die Elemente von A und B zyklentreu aufeinander ab, woraus die Gleichheit der Zyklenindizes von σ und $\widetilde{\sigma}$ und somit die erste Aussage des Satzes folgt.

Seien nun $\alpha \in S_A$ und $\beta \in S_B$ mit dem selben Zyklenindex $(k_1, k_2, \ldots, k_n), \sum_{i=1}^n i \cdot k_i = n$, dann existieren Familien $(A_{i,j})$ und $(B_{i,j})$ mit $(i,j) \in I := \{(a,b) \in \mathbb{N} \times \mathbb{N} : a \leq n \wedge b \leq k_a\}$ sodass $A_{i,j}$ bzw $B_{i,j}$ genau die Elemente jeweils eines Zyklus der Länge i von α bzw. β enthält, $(i_1, j_1) \neq (i_2, j_2) \Rightarrow A_{i_1,j_1} \cap A_{i_2,j_2} = \emptyset \wedge B_{i_1,j_1} \cap B_{i_2,j_2} = \emptyset$ für alle $(i_1, j_1), (i_2, j_2) \in I$, und $\bigcup_{(i,j)\in I} A_{i,j} = A$ sowie $\bigcup_{(i,j)\in I} B_{i,j} = B$. Weiters existieren Familien $(a_{i,j})$ und $(b_{i,j})$ mit $(i,j) \in I$ und $a_{i,j} \in A_{i,j}$, $b_{i,j} \in B_{i,j}$ für alle $(i,j) \in I$. Betrachten wir nun die Menge

$$K := \left\{(i,j,n) \in (\mathbb{N}_0)^3 \,:\, (i,j) \in I \wedge 0 \leq n < i\right\},$$

dann ist die Abbildung $\Phi : K \to A, (i,j,n) \mapsto \alpha^n(a_{i,j})$ eine Bijektion. Φ ist injektiv, denn es gilt für alle $(i_1, j_1, n_1), (i_2, j_2, n_2) \in K$

$$\Phi(i_1, j_1, n_1) = \Phi(i_2, j_2, n_2) \Rightarrow \alpha^{n_1}(a_{i_1,j_1}) = \alpha^{n_2}(a_{i_2,j_2}).$$

Da für alle $n \in \mathbb{N}_0$ und alle $(i,j) \in I$ gilt, dass $\alpha^n(a_{i,j}) \in A_{i,j}$, und weil die Mengen $A_{i,j}$ paarweise disjunkt sind, muss also $(i_1, j_1) = (i_2, j_2)$ gelten. Angenommen, $n_1 \neq n_2$, o.B.d.A $n_1 < n_2$, dann gilt $a_{i_1,j_1} = \alpha^{n_2-n_1}(a_{i_1,j_1})$ mit $n_2 - n_1 < i$, im Widerspruch zur Zyklenlänge i. Es gilt also auch $n_1 = n_2$. Die Menge K besteht genau aus $\sum_{i=1}^n i \cdot k_i = n$ Elementen, es gilt also $|K| = |A|$, woraus aufgrund der Injektivität von Φ auch die Surjektivität und somit die Bijektivität folgt. Analoges gilt für die Abbildung

$$\Psi : K \to B, (i,j,n) \mapsto \beta^n(b_{i,j}).$$

Schließlich zeigen wir nun, dass die Bijektion $\tau := \Psi \circ \Phi^{-1}$ das Gewünschte leistet. Dabei werden wir verwenden, dass für alle $(i,j) \in I$ wegen der Zyklenlänge i sowohl $\alpha^i(a_{i,j}) = a_{i,j}$, als auch $\beta^i(b_{i,j}) = b_{i,j}$ gilt. Weiters definieren wir $\Phi^{-1}(a) =: (i_a, j_a, n_a)$ für alle $a \in A$ sowie $\Psi^{-1}(b) =: (i_b, j_b, n_b)$ für alle $b \in B$. Somit gilt $a = (\Phi \circ \Phi^{-1})(a) = \alpha^{n_a}(a_{i_a,j_a})$ und $b = (\Psi \circ \Psi^{-1})(b) = \beta^{n_b}(b_{i_b,j_b})$. Wir müssen zeigen, dass $\tau \circ \alpha \circ \tau^{-1}(b) = \Psi \circ \Phi^{-1} \circ \alpha \circ \Phi \circ \Psi^{-1}(b) = \beta(b)$ für alle $b \in B$ ist. Dazu gehen wir schrittweise vor und zeigen zuerst, dass für alle $(i,j,n) \in K$ gilt, dass

$$\begin{aligned}\Phi^{-1} \circ \alpha \circ \Phi(i,j,n) &= \Phi^{-1} \circ \alpha(\alpha^n(a_{i,j})) = \Phi^{-1}(\alpha^{n+1}(a_{i,j}))\\ &= \begin{cases}\Phi^{-1}(\alpha^0(a_{i,j})) & \text{falls } n = i-1,\\ \Phi^{-1}(\alpha^{n+1}(a_{i,j})) & \text{falls } n < i-1\end{cases} = \begin{cases}(i,j,0) & \text{falls } n = i-1,\\ (i,j,n+1) & \text{falls } n < i-1.\end{cases}\end{aligned}$$

Nun gilt für alle $b \in B$:

$$\begin{aligned}\tau \circ \alpha \circ \tau^{-1}(b) &= \Psi \circ \Phi^{-1} \circ \alpha \circ \Phi \circ \Psi^{-1}(b) = \Psi \circ \Phi^{-1} \circ \alpha \circ \Phi(i_b, j_b, n_b) \\ &= \begin{cases} \Psi(i_b, j_b, 0) & \text{falls } n_b = i-1, \\ \Psi(i_b, j_b, n_b+1) & \text{falls } n_b < i-1 \end{cases} \\ &= \begin{cases} \beta^0(b_{i_b,j_b}) = \beta^i(b_{i_b,j_b}) = \beta^{n_b+1}(b_{i_b,j_b}) & \text{falls } n_b = i-1, \\ \beta^{n_b+1}(b_{i_b,j_b}) & \text{falls } n_b < i-1 \end{cases} \\ &= \beta^{n_b+1}(b_{i_b,j_b}) = \beta(\beta^{n_b}(b_{i_b,j_b})) = \beta(b),\end{aligned}$$

was zu zeigen war. Für den Fall, dass $n = 0$ ist, gilt $A = B = \emptyset$ und es gibt genau die leere Abbildung $f_\emptyset$. Der Satz ist somit trivialerweise erfüllt. □

Lemma 2.3.7. *Sei $\mathcal{F}$ eine kombinatorische Spezies und $(U_n)_{n\in\mathbb{N}_0}$ und $(V_n)_{n\in\mathbb{N}_0}$ Folgen von endlichen Mengen mit $|U_n| = |V_n| = n$ für alle $n \in \mathbb{N}_0$, so gilt*

$$\sum_{n\geq 0} \frac{1}{n!} \sum_{\sigma\in S_{U_n}} \mathit{fix}\, \mathcal{F}[\sigma] \cdot z_1^{\sigma_1} z_2^{\sigma_2} \cdots z_n^{\sigma_n} = \sum_{n\geq 0} \frac{1}{n!} \sum_{\sigma\in S_{V_n}} \mathit{fix}\, \mathcal{F}[\sigma] \cdot z_1^{\sigma_1} z_2^{\sigma_2} \cdots z_n^{\sigma_n},$$

wobei S_{U_n} und S_{V_n} die symmetrischen Gruppen auf den Mengen U_n bzw. V_n bezeichnen.

Beweis. Da für alle $n \in \mathbb{N}_0$ die Mengen U_n und V_n die gleiche Mächtigkeit besitzen, existiert eine Familie von Bijektionen $(\tau_n : U_n \to V_n)_{n\in\mathbb{N}_0}$. Nun zeigen wir zuerst, dass für $n \in \mathbb{N}_0$ die Abbildung

$$\Phi_n : S_{U_n} \to S_{V_n}, \sigma \mapsto \tau \circ \sigma \circ \tau^{-1}$$

ein Gruppenisomorphismus ist. Für $\sigma_1, \sigma_2 \in S_{U_n}$ gilt

$$\begin{aligned}&\Phi_n(\sigma_1) = \Phi_n(\sigma_2) \Leftrightarrow \tau_n \circ \sigma_1 \circ \tau_n^{-1} = \tau_n \circ \sigma_2 \circ \tau_n^{-1} \\ \Leftrightarrow\quad &\tau_n^{-1} \circ \tau_n \circ \sigma_1 \circ \tau_n^{-1} \circ \tau_n = \tau_n^{-1} \circ \tau_n \circ \sigma_2 \circ \tau_n^{-1} \circ \tau_n \Leftrightarrow \sigma_1 = \sigma_2.\end{aligned}$$

Daher ist Φ_n injektiv. Für $\rho \in S_{V_n}$ gilt mit $\sigma := \tau_n^{-1} \circ \rho \circ \tau_n$, dass

$$\Phi_n(\sigma) = \tau_n \circ \sigma \circ \tau_n^{-1} = \tau_n \circ \tau_n^{-1} \circ \rho \circ \tau_n \circ \tau_n^{-1} = \rho.$$

Φ_n ist also auch surjektiv und daher bijektiv. Schließlich gilt für $\sigma_1, \sigma_2 \in S_{U_n}$, dass

$$\begin{aligned}\Phi_n(\sigma_1 \circ \sigma_2) &= \tau_n \circ \sigma_1 \circ \sigma_2 \circ \tau_n^{-1} = \tau_n \circ \sigma_1 \circ \mathrm{id}_{U_n} \circ \sigma_2 \circ \tau_n^{-1} \\ &= \tau_n \circ \sigma_1 \circ \tau_n^{-1} \circ \tau \circ \sigma_2 \circ \tau_n^{-1} = \Phi_n(\sigma_1) \circ \Phi_n(\sigma_2).\end{aligned}$$

Die Homomorphiebedingung ist also ebenfalls erfüllt und Φ_n ist ein Gruppenisomorphismus. Aus Lemma 2.3.6 folgt außerdem, dass σ und $\Phi_n(\sigma)$ für alle $\sigma \in S_{U_n}$ den selben Zyklenindex haben.

Nun zeigen wir, dass für alle $n \in \mathbb{N}_0$ und alle $\sigma \in S_{U_n}$ gilt, dass fix $\mathcal{F}[\sigma]$ = fix $\mathcal{F}[\Phi_n(\sigma)]$. Wir sehen, dass

$$\begin{aligned}\mathcal{F}[\sigma](s) = s &\Leftrightarrow \mathcal{F}[\tau_n](\mathcal{F}[\sigma](s)) = \mathcal{F}[\tau_n](s) \Leftrightarrow \mathcal{F}[\tau_n]\big(\mathcal{F}[\sigma](\mathcal{F}[\mathrm{id}](s))\big) = \mathcal{F}[\tau_n](s) \\ &\Leftrightarrow \mathcal{F}[\tau_n]\big(\mathcal{F}[\sigma](\mathcal{F}[\tau_n^{-1} \circ \tau_n](s))\big) = \mathcal{F}[\tau_n](s) \\ &\Leftrightarrow \mathcal{F}[\tau_n]\big(\mathcal{F}[\sigma](\mathcal{F}[\tau_n^{-1}](\mathcal{F}[\tau_n](s)))\big) = \mathcal{F}[\tau_n](s) \\ &\Leftrightarrow \mathcal{F}[\tau_n \circ \sigma \circ \tau_n^{-1}](\mathcal{F}[\tau_n](s)) = \mathcal{F}[\tau_n](s) \\ &\Leftrightarrow \mathcal{F}[\Phi_n(\sigma)](\mathcal{F}[\tau_n](s)) = \mathcal{F}[\tau_n](s).\end{aligned}$$

Die Abbildung $\mathcal{F}[\tau_n]\big|_{\text{Fix } \mathcal{F}[\sigma]}$ ist also eine Bijektion zwischen Fix $\mathcal{F}[\sigma]$ und Fix $\mathcal{F}[\Phi(\sigma)]$, woraus die Gleichheit der Anzahl der Fixpunkte folgt. Abschließend gilt daher

$$\begin{aligned}
&\sum_{n\geq 0}\frac{1}{n!}\sum_{\sigma\in S_{U_n}}\text{fix }\mathcal{F}[\sigma]z_1^{\sigma_1}z_2^{\sigma_2}\cdots z_n^{\sigma_n}\\
=&\sum_{n\geq 0}\frac{1}{n!}\sum_{\sigma\in S_{U_n}}\text{fix }\mathcal{F}[\Phi(\sigma)]z_1^{\Phi(\sigma)_1}z_2^{\Phi(\sigma)_2}\cdots z_n^{\Phi(\sigma)_n}\\
=&\sum_{n\geq 0}\frac{1}{n!}\sum_{\rho\in S_{V_n}}\text{fix }\mathcal{F}[\rho]z_1^{\rho_1}z_2^{\rho_2}\cdots z_n^{\rho_n}.
\end{aligned}$$

□

Nun werden wir untersuchen, wie sich die Anzahl der Strukturen, die durch den Transport von Strukturen festgehalten werden, verhält, wenn man Permutationen mit gleicher Zyklenstruktur betrachtet. Wie wir sehen werden, kann die Zyklenindexreihe noch erheblich vereinfacht werden.

Lemma 2.3.8. *Sei $\mathcal{F}$ eine kombinatorische Spezies, $n\in\mathbb{N}_0$ und $\sigma_1,\sigma_2\in S_n$ mit dem selben Zyklenindex $(k_1,k_2,\ldots,k_n)$, $\sum_{i=1}^n i\cdot k_i=n$, dann gilt fix $\mathcal{F}[\sigma_1]=$ fix $\mathcal{F}[\sigma_2]$.*

Beweis. Für $\sigma_1,\sigma_2\in S_n$ mit selbem Zyklenindex $(k_1,k_2,\ldots,k_n)$ existiert aufgrund des Lemmas 2.3.8 eine Bijektion $\tau:[n]\to[n]$ mit $\sigma_2=\tau\circ\sigma_1\circ\tau^{-1}$. Nun gilt für alle $s\in\mathcal{F}[n]$

$$\begin{aligned}
\mathcal{F}[\sigma_1](s)=s &\Leftrightarrow \mathcal{F}[\tau](\mathcal{F}[\sigma_1](s))=F[\tau](s)\Leftrightarrow \mathcal{F}[\tau](\mathcal{F}[\sigma_1](\mathcal{F}[\text{id}](s)))=F[\tau](s)\\
&\Leftrightarrow \mathcal{F}[\tau](\mathcal{F}[\sigma_1](\mathcal{F}[\tau^{-1}\circ\tau](s)))=F[\tau](s)\Leftrightarrow \mathcal{F}[\tau](\mathcal{F}[\sigma_1](\mathcal{F}[\tau^{-1}](\mathcal{F}[\tau](s))))=F[\tau](s)\\
&\Leftrightarrow \mathcal{F}[\tau\circ\sigma_1\circ\tau^{-1}](\mathcal{F}[\tau](s))=F[\tau](s)\Leftrightarrow \mathcal{F}[\sigma_2](\mathcal{F}[\tau](s))=F[\tau](s).
\end{aligned}$$

Die Fixpunkte von $\mathcal{F}[\sigma_1]$ werden durch $\mathcal{F}[\tau]$ also bijektiv auf die Fixpunkte von $\mathcal{F}[\sigma_2]$ abgebildet, woraus fix $\mathcal{F}[\sigma_1]=$ fix $\mathcal{F}[\sigma_2]$ folgt. □

Dieses Lemma führt zu der folgenden Definition, die aufgrund der Invarianz der Anzahl der Fixpunkte des Transports der Strukturen bezüglich Permutationen mit gleichem Zyklenindex sinnvoll ist.

Definition 2.3.9. Sei $\mathcal{F}$ eine kombinatorische Spezies und $(k_1,k_2,\ldots,k_n),\sum_{i=1}^n i\cdot k_i=n$ ein beliebiger Zyklenindex. Wir definieren fix $\mathcal{F}[k_1,k_2,\ldots,k_n]:=$ fix $\mathcal{F}[\sigma]$, wobei σ eine beliebige Permutation aus S_n mit dem Zyklenindex $(k_1,k_2,\ldots,k_n)$ ist.

Für die angestrebte Vereinfachung der Zyklenindexreihe benötigen wir nun noch die Anzahl der Permutationen, die zu einem gegebenen Zyklenindex existieren. Die Antwort auf die Frage liefert das nächste Lemma.

Lemma 2.3.10. *Sei $n\in\mathbb{N}_0$ und $k:=(k_1,k_2,\ldots,k_n),\sum_{i=1}^n k_i=n$ ein Zyklentyp, dann existieren genau*

$$\frac{n!}{aut(k_1,k_2,\ldots,k_n)} \tag{11}$$

Permutationen aus S_n mit dem Zyklentyp k, wobei

$$aut(k_1,k_2,\ldots,k_n):=k_1!1^{k_1}k_2!2^{k_2}\cdots k_n!n^{k_n} \tag{12}$$

gilt.

Beweis. Jeder Permutation $\sigma\in S_n$ vom Zyklenindex $(k_1,k_2,\ldots,k_n)$ ist in umkehrbar eindeutiger weise eine Menge von disjunkten Zyklen zugeordnet, die aus genau k_1 einelementigen, k_2 zweielementigen, …, k_n n-elementigen Zyklen besteht, und sodass jedes Element $i\in[n]$ in genau einem Zyklus der Menge vorkommt. Um nun eine Permutation vom Zyklenindex $(k_1,k_2,\ldots,k_n)$

zu konstruieren, bilden wir zuerst eine Partition der Menge $[n]$ aus k_1 einelementigen, k_2 zweielementigen, $\ldots$, k_n n-elementigen Teilmengen. Dann erzeugen wir für jedes Element dieser Partition einen Zyklus, und fassen die Zyklen dann zu einer Menge der eingangs erwähnten Gestalt zusammen. Es ist leicht zu sehen, dass man in dieser Weise jede Permutation aus S_n mit dem gegebenen Zyklenindex konstruieren kann.

Die Anzahl der Partitionen der Menge $[n]$ von obiger Gestalt ist

$$\frac{n!}{(1!)^{k_1}\cdot(2!)^{k_2}\cdots(n!)^{k_n}}\cdot\frac{1}{k_1!k_2!\cdot k_n!}.$$

Aus jeder beliebigen i-elementiger Menge aus der Partition können $\frac{i!}{i}$ Zyklen mit den Elementen der Menge geformt werden. Insgesamt erhalten wir also:

$$\begin{aligned}
&|\{\sigma\in S_n : \text{Zyklenindex von } \sigma=(k_1,k_2,\ldots,k_n)\}| = \\
&= \frac{n!}{(1!)^{k_1}(2!)^{k_2}\cdots(n!)^{k^n}}\cdot\frac{1}{k_1!k_2!\cdot k_n!}\cdot\left(\frac{1!}{1}\right)^{k_1}\left(\frac{2!}{2}\right)^{k_2}\cdots\left(\frac{n!}{n}\right)^{k_n} \\
&= \frac{n!}{1^{k_1}k_1!2^{k_2}k_2!\cdots n^{k_n}k_n!} = \frac{n!}{\mathrm{aut}(k_1,k_2,\ldots,k_n)}.
\end{aligned}$$

□

Nun kommen wir zur erwähnten Vereinfachung der Zyklenindexreihe.

Satz 2.3.11. *Sei $\mathcal{F}$ eine kombinatorische Spezies, dann gilt für ihre Zyklenindexreihe*

$$Z_F(z_1,z_2,\ldots)=\sum_{n\geq 0}\sum_{k_1+2k_2+\ldots+nk_n=n}\frac{\mathit{fix}\,\mathcal{F}[k_1,k_2,\ldots,k_n]}{\mathit{aut}(k_1,k_2,\ldots,k_n)}z_1^{k_1}z_2^{k_2}\cdots z_n^{k_n}.$$

Beweis. Es gilt

$$\begin{aligned}
Z_F(z_1,z_2,\ldots) &= \sum_{n\geq 0}\frac{1}{n!}\sum_{\sigma\in S_n}\mathrm{fix}\,\mathcal{F}[\sigma]z_1^{\sigma_1}z_2^{\sigma_2}\cdots z_n^{\sigma_n} \\
&= \sum_{n\geq 0}\frac{1}{n!}\sum_{k_1+2k_2+\ldots nk_n=n}\sum_{\substack{\sigma\in S_n\\(\sigma_1,\sigma_2,\ldots,\sigma_k)=\\(k_1,k_2,\ldots,k_n)}}\mathrm{fix}\,\mathcal{F}[\sigma]z_1^{\sigma_1}z_2^{\sigma_2}\cdots z_n^{\sigma_n} \\
&\overset{\text{Lemma 2.3.8}}{=} \sum_{n\geq 0}\frac{1}{n!}\sum_{k_1+2k_2+\ldots nk_n=n}\sum_{\substack{\sigma\in S_n\\(\sigma_1,\sigma_2,\ldots,\sigma_k)=\\(k_1,k_2,\ldots,k_n)}}\mathrm{fix}\,\mathcal{F}[k_1,k_2,\ldots,k_n]z_1^{\sigma_1}z_2^{\sigma_2}\cdots z_n^{\sigma_n} \\
&= \sum_{n\geq 0}\frac{1}{n!}\sum_{k_1+2k_2+\ldots nk_n=n}\frac{n!}{\mathrm{aut}(k_1,k_2,\ldots,k_n)}\mathrm{fix}\,\mathcal{F}[k_1,k_2,\ldots,k_n]z_1^{k_1}z_2^{k_n}\cdots z_n^{k_n} \\
&= \sum_{n\geq 0}\sum_{k_1+2k_2+\ldots+nk_n=n}\frac{\mathrm{fix}\,\mathcal{F}[k_1,k_2,\ldots,k_n]}{\mathrm{aut}(k_1,k_2,\ldots,k_n)}z_1^{k_1}z_2^{k_2}\cdots z_n^{k_n}.
\end{aligned}$$

□

In der Zyklenindexreihe einer Spezies ist die gesamte Information enthalten, um daraus die exponentiell und die gewöhnliche erzeugende Funktion ableiten zu können. Um die Herleitungen zu beweisen, müssen wir zuerst ein paar Begriffe aus dem Bereich der Gruppenaktionen einführen.

Definition 2.3.12. Sei $(G,\circ)$ eine Gruppe und M eine beliebige Menge, dann nennt man eine Abbildung $\alpha:(G,M)\to M$ eine *Gruppenaktion* der Gruppe G auf der Menge M genau dann, wenn mit der Schreibweise $g\cdot m:=\alpha(g,m)$

1. für alle $g, h \in G, m \in M$ gilt, dass $g \cdot (h \cdot m) = (g \circ h) \cdot m$, und
2. für alle $m \in M$ gilt, dass $e \cdot m = m$, wobei e das neutrale Element der Gruppe G bezeichnet.

Lemma 2.3.13. *Sei $\alpha : (G, M) \to M$ eine Gruppenaktion, dann ist die Abbildung*

$$\phi_\alpha : G \to S_M, g \mapsto (m \mapsto g \cdot m)$$

ein Gruppenhomomorphismus. Er heißt die durch α induzierte Permutationsrepräsentation von G.

Beweis. Zuerst zeigen wir, dass $\phi_\alpha(g)$ für alle $g \in G$ tatsächlich in S_M liegt. Sei $g \in G$ und $m_1, m_2 \in M$, so gilt

$$\begin{aligned}\phi_\alpha(g)(m_1) = \phi_\alpha(g)(m_2) &\Rightarrow g \cdot m_1 = g \cdot m_2 \Rightarrow g^{-1} \cdot (g \cdot m_1) = g^{-1} \cdot (g \cdot m_2) \\ &\Rightarrow (g^{-1}g) \cdot m_1 = (g^{-1}g) \cdot m_2 \Rightarrow e \cdot m_1 = e \cdot m_2 \Rightarrow m_1 = m_2.\end{aligned}$$

$\phi_\alpha(g)$ ist also injektiv. Sei nun $m \in M$, so gilt

$$\phi_\alpha(g)(g^{-1} \cdot m) = g \cdot (g^{-1} \cdot m) = (gg^{-1}) \cdot m = e \cdot m = m.$$

$\phi_\alpha(g)$ ist also surjektiv und somit insgesamt bijektiv und daher ein Element aus S_M. Dass die Homomorphiebedingung erfüllt ist, sehen wir leicht, da für alle $g_1, g_2 \in G$ und alle $m \in M$ gilt, dass

$$\begin{aligned}\phi_\alpha(g_1g_2)(m) &= (g_1g_2) \cdot m = g_1 \cdot (g_2 \cdot m) = \phi_\alpha(g_1)(g_2 \cdot m) \\ &= \phi_\alpha(g_1)(\phi_\alpha(g_2)(m)) = \big(\phi_\alpha(g_1) \circ \phi_\alpha(g_2)\big)(m).\end{aligned}$$

□

Lemma 2.3.14. *Jede Gruppenaktion $\alpha : (G, M) \to M$ induziert auf der Menge M eine Äquivalenzrelation $\sim_G$, die so definiert ist, dass Elemente $x, y \in M$ genau dann in Relation stehen, wenn ein $g \in G$ existiert, sodass $y = g \cdot x$. Die Äquivalenzklasssen werden* Orbits *genannt. Sei $x \in M$, dann wird die Äquivalenzklasse von x bezüglich $\sim_G$ als* Orbit von x*, auch als $\mathcal{O}(x)$ geschrieben, bezeichnet, und es gilt*

$$\mathcal{O}(x) = \{y \in M \,:\, \exists g \in G : y = g \cdot x\}.$$

Die Menge der Orbits $\{\mathcal{O}(x) \,:\, x \in M\}$ wird auch als M/G bezeichnet.

Beweis. Für alle $x \in M$ und das neutrale Element $e \in G$ gilt $e \cdot x = x$, also ist $\sim_G$ reflexiv. Stehen x und y in Relation zueinander, so existiert ein $g \in G$, sodass $y = g \cdot x$. Also gilt auch $g^{-1} \cdot y = g^{-1} \cdot (g \cdot x) = (g^{-1}g) \cdot x = e \cdot x = x$, und somit $y \sim_G x$. Die Relation ist also auch symmetrisch. Gilt $x \sim_G y$ und $y \sim_G z$, so existierten $g, h \in G$, sodass $y = g \cdot x$ und $z = h \cdot y$. Daraus folgt $z = h \cdot (g \cdot x) = (hg) \cdot x$. Es gilt also auch $x \sim_G z$ und die Relation ist somit transitiv und insgesamt eine Äquivalenzrelation. Für die Orbits von $x \in M$ gilt

$$\mathcal{O}(x) = \{y \in M \,:\, \exists g \in G : y = g \cdot x\} = \{y \in M \,:\, x \sim_G y\} = [x]_{\sim_G}.$$

Es handelt sich hierbei also um die Äquivalenzklassen von $\sim_G$. □

Für die angestrebte Herleitung der exponentiell und gewöhnlichen erzeugenden Funktionen aus den Zyklenindexreihen benötigen wir noch folgendes Lemma, dessen Beweis in [2, S. 395-397] zu finden ist.

Satz 2.3.15 (Lemma von Burnside)**.** *Die Anzahl der Orbits einer Gruppenaktion einer endlichen Gruppe G auf einer endlichen Menge M beträgt*

$$|M/G| = \frac{1}{|G|} \sum_{g \in G} |Fix\,(\phi(g))|\,, \tag{13}$$

wobei ϕ die durch die Gruppenaktion induzierte Permutationsdarstellung und Fix $(\phi(g))$ die Menge der Fixpunkte von $\phi(g)$ bezeichnet.

Im Kontext der kombinatorischen Spezies interessieren uns nun die Gruppenaktionen, die mittels des Transports der Strukturen definiert werden können.

Lemma 2.3.16. *Sei $\mathcal{F}$ eine kombinatorische Spezies, U eine endliche Menge und S_U die symmetrische Gruppe auf U, dann ist die Abbildung*

$$\gamma_{\mathcal{F},U} : (S_U, \mathcal{F}[U]) \to \mathcal{F}[U], (\sigma, s) \to \mathcal{F}[\sigma](s)$$

eine Gruppenaktion der Gruppe S_U auf der Menge $\mathcal{F}[U]$ der $\mathcal{F}$-Strukturen der Menge U. Die Orbits dieser Gruppenaktion sind genau die Isomorphietypen der $\mathcal{F}$-Strukturen auf U, und für die induzierte Permutationsdarstellung gilt $\phi_{\gamma_{\mathcal{F},U}}(\sigma) = \mathcal{F}[\sigma]$ für alle $\sigma \in S_U$.

Beweis. Es gilt für alle $\sigma_1, \sigma_2 \in S_U$ und alle $s \in \mathcal{F}[U]$

$$\sigma_1 \cdot (\sigma_2 \cdot s) = \sigma_1 \cdot (\mathcal{F}[\sigma_2](s)) = \mathcal{F}[\sigma_1]\big(\mathcal{F}[\sigma_2](s)\big) = \mathcal{F}[\sigma_1 \circ \sigma_2](s) = (\sigma_1 \circ \sigma_2) \cdot s.$$

Weiters gilt für die identische Abbildung $\mathrm{id} \in S_U$ und für alle $s \in \mathcal{F}[U]$

$$\mathrm{id} \cdot s = \mathcal{F}[\mathrm{id}](s) = s.$$

Sei nun $s \in \mathcal{F}[U]$, dann gilt

$$\mathcal{O}(s) = \{t \in \mathcal{F}[U] \,:\, \exists \sigma \in S_U : t = \sigma \cdot s\} = \{t \in \mathcal{F}[U] \,:\, \exists \sigma \in S_U : t = \mathcal{F}[\sigma](s)\}.$$

Die Orbits der Gruppenaktion stimmen also mit den Isomorphietypen überein. Schließlich gilt für alle $s \in \mathcal{F}[U]$ und alle $\sigma \in S_U$

$$\phi_{\gamma_{\mathcal{F},U}}(\sigma)(s) = \sigma \cdot s = \mathcal{F}[\sigma](s),$$

woraus die letzte Behauptung des Lemmas folgt. □

Nun kommen wir zum Hauptergebnis dieses Abschnitts, der angesprochenen Ableitung der exponentiell und der gewöhnlichen erzeugenden Funktionen aus den Zyklenindexreihen.

Satz 2.3.17. *Sei $\mathcal{F}$ eine kombinatorische Spezies und $Z_F(z_1, z_2, \ldots)$ die ihr zugeordnete Zyklenindexreihe, dann gilt für ihre exponentiell erzeugende Funktion $F(z)$ und ihre gewöhnliche erzeugende Funktion $\widetilde{F}(z)$*

$$F(z) = Z_F(z, 0, 0, \ldots), \text{ und}$$
$$\widetilde{F}(z) = Z_F(z, z^2, z^3, \ldots).$$

Beweis. Zuerst beweisen wir die Identität für die exponentiell erzeugende Funktion. Wenn wir bedenken, dass das Monom $z^{\sigma_1}0^{\sigma_2}\cdots 0^{\sigma_n}$ genau dann ungleich 0 ist, wenn $\sigma_i = 0$ für alle $i \neq 1$, die einzige Permutation aus S_n mit Zyklenindex $(n, 0, \ldots, 0)$ die Identität ist, und der Transport der Strukturen entlang der Identität alle Strukturen festhält, sehen wir

$$Z_F(z, 0, 0, \ldots) = \sum_{n\geq 0} \frac{1}{n!} \sum_{\sigma \in S_n} \mathrm{fix}\ \mathcal{F}[\sigma] z^{\sigma_1} 0^{\sigma_2} \cdots 0^{\sigma_n} = \sum_{n\geq 0} \frac{1}{n!} \sum_{\substack{\sigma \in S_n \\ (\sigma_1, \sigma_2, \ldots, \sigma_n) = \\ (n, 0, 0, \ldots)}} \mathrm{fix}\ \mathcal{F}[\sigma] z^n$$

$$= \sum_{n\geq 0} \frac{1}{n!} \mathrm{fix}\ \mathcal{F}[\mathrm{id}] z^n = \sum_{n\geq 0} \frac{|\mathcal{F}[n]|}{n!} z^n = F(z).$$

Nun wenden wir uns der gewöhnlichen erzeugenden Funktion zu. Es gilt

$$\widetilde{F}(z, z^2, z^3, \ldots) = \sum_{n\geq 0} \frac{1}{n!} \sum_{\sigma \in S_n} \mathrm{fix}\ \mathcal{F}[\sigma](z)^{\sigma_1}(z^2)^{\sigma_2}\cdots(z^n)^{\sigma_n} = \sum_{n\geq 0} \frac{1}{n!} \sum_{\sigma \in S_n} \mathrm{fix}\ \mathcal{F}[\sigma] z^{\sigma_1} z^{2\sigma_2} \cdots z^{n\sigma_n}$$

$$= \sum_{n\geq 0} \frac{1}{n!} \sum_{\sigma \in S_n} \mathrm{fix}\ \mathcal{F}[\sigma] z^n \overset{\text{Lemma 2.3.16}}{=} \sum_{n\geq 0} \frac{1}{|S_n|} \sum_{\sigma \in S_n} \left|\mathrm{Fix}\ \phi_{\gamma_{\mathcal{F},[n]}}(\sigma)\right| z^n$$

$$\overset{\text{Lemma von Burnside}}{=} \sum_{n\geq 0} |\mathcal{F}[n]/S_n|\, z^n \overset{\text{Lemma 2.3.16}}{=} \sum_{n\geq 0} |T(\mathcal{F}, n)|\, z^n = \widetilde{F}(z).$$

□

Nun zeigen wir noch, dass für isomorphe Spezies $\mathcal{F}$ und $\mathcal{G}$ auch deren erzeugende Funktionen übereinstimmen.

Satz 2.3.18. *Seien $\mathcal{F}$ und $\mathcal{G}$ zwei isomorphe kombinatorische Spezies, dann gilt für deren exponentiell erzeugenden Funktionen $F(z), G(z)$, deren gewöhnlichen erzeugenden Funktionen $\widetilde{F}(z), \widetilde{G}(z)$ und deren Zyklenindexreihen $Z_F(z_1, z_2, \ldots), Z_G(z_1, z_2, \ldots)$*

$$\begin{aligned} F(z) &= G(z) \\ \widetilde{F}(z) &= \widetilde{G}(z) \\ Z_F(z_1, z_2, \ldots) &= Z_G(z_1, z_2, \ldots). \end{aligned}$$

Beweis. Da die Spezies $\mathcal{F}$ und $\mathcal{G}$ isomorph sind, existiert eine mit den endlichen Mengen indizierte Familie von Bijektionen $\alpha_U : \mathcal{F}[U] \to \mathcal{G}[U]$, sodass für alle endlichen Mengen U und V mit $|U| = |V|$, alle Bijektionen $\sigma : U \to V$ und alle $\mathcal{F}$-Strukturen s auf der Menge U gilt, dass

$$\mathcal{G}[\sigma](\alpha_U(s)) = \alpha_V(\mathcal{F}[\sigma](s)).$$

Nun gilt für jede endliche Menge U, jede Permutation $\sigma \in S_U$ und jede $\mathcal{F}$-Struktur $s \in \mathcal{F}[U]$, dass

$$\mathcal{F}[\sigma](s) = s \Leftrightarrow \alpha_U\big(\mathcal{F}[\sigma](s)\big) = \alpha_U(s) \Leftrightarrow \mathcal{G}[\sigma]\big(\alpha_U(s)\big) = \alpha_U(s).$$

Die Fixpunkte von $\mathcal{F}[\sigma]$ werden durch α_U also bijektiv auf die Fixpunkte von $\mathcal{G}[\sigma]$ abgebildet, woraus

$$\operatorname{fix} \mathcal{F}[\sigma] = \operatorname{fix} \mathcal{G}[\sigma]$$

folgt. Für die Zyklenindexreihe bedeutet das also:

$$\begin{aligned} Z_F(z_1, z_2, \ldots) &= \sum_{n \geq 0} \frac{1}{n!} \sum_{\sigma \in S_n} \operatorname{fix} \mathcal{F}[\sigma] z_1^{\sigma_1} z_2^{\sigma_2} \cdots z_n^{\sigma_n} \\ &= \sum_{n \geq 0} \frac{1}{n!} \sum_{\sigma \in S_n} \operatorname{fix} \mathcal{G}[\sigma] z_1^{\sigma_1} z_2^{\sigma_2} \cdots z_n^{\sigma_n} = Z_G(z_1, z_2, \ldots). \end{aligned}$$

Die Identitäten für die exponentiell bzw. gewöhnlichen erzeugenden Funktionen folgen aus Satz 2.3.17. □

Abschließend sind in Tabelle 2.3 noch die exponentiellen und gewöhnlichen erzeugenden Funktionen beziehungsweise die Zyklenindexreihen der in Definition 2.2.2 angegebenen Spezies zusammengefasst. Den Beweis der Identitäten findet man beispielsweise in [3, S. 8]. Die Funktion ϕ bezeichnet die *eulersche Phi-Funktion*, die als

$$\phi : \mathbb{N} \to \mathbb{N}, n \mapsto \left|\left\{a \in \mathbb{N} \,:\, 1 \leq a \leq n \wedge \operatorname{ggT}(a, n) = 1\right\}\right| \tag{14}$$

definiert ist, wobei $\operatorname{ggT}(a, n)$ den größten gemeinsamen Teiler von a und n bezeichnet.

$\mathcal{F}$	$Z_F(z_1, z_2)$	$F(z)$	$\widetilde{F}(z)$
$\mathbf{0}$	$Z_{\mathbf{0}}(z_1, z_2, \ldots) = 0$	$\mathbf{0}(z) = 0$	$\widetilde{\mathbf{0}}(z) = 0$
$\mathbf{1}$	$Z_{\mathbf{1}}(z_1, z_2, \ldots) = 1$	$\mathbf{1}(z) = 1$	$\widetilde{\mathbf{1}}(z) = 1$
$\mathcal{Z}$	$Z_Z(z_1, z_2, \ldots) = z_1$	$Z(z) = z$	$\widetilde{Z}(z) = z$
SET	$Z_{\mathrm{SET}}(z_1, z_2, \ldots) = \exp\left(\sum_{k=1}^{\infty} \frac{z_k}{k}\right)$	$\mathrm{SET}(z) = \exp(z)$	$\widetilde{\mathrm{SET}}(z) = \exp\left(\sum_{k=1}^{\infty} \frac{z^k}{k}\right)$
SEQ	$Z_{\mathrm{SEQ}}(z_1, z_2, \ldots) = \frac{1}{1 - z_1}$	$\mathrm{SEQ}(z) = \frac{1}{1 - z}$	$\widetilde{\mathrm{SEQ}}(z) = \frac{1}{1 - z}$
$\mathcal{P}$	$Z_{\mathcal{P}}(z_1, z_2, \ldots) = \prod_{k \geq 1} \frac{1}{1 - z_k}$	$\mathcal{P}(z) = \frac{1}{1 - z}$	$\widetilde{\mathcal{P}}(z) = \prod_{k \geq 1} \frac{1}{1 - z^k}$
CYC	$Z_{\mathrm{CYC}}(z_1, z_2, \ldots) = \sum_{k \geq 1} \frac{\phi(k)}{k} \log \frac{1}{1 - z_k}$	$\mathrm{CYC}(z) = \log \frac{1}{1 - z}$	$\widetilde{\mathrm{CYC}}(z) = \sum_{k \geq 1} \frac{\phi(k)}{k} \log \frac{1}{1 - z^k}$

Tabelle 1: Erzeugende Funktionen der grundlegenden Spezies

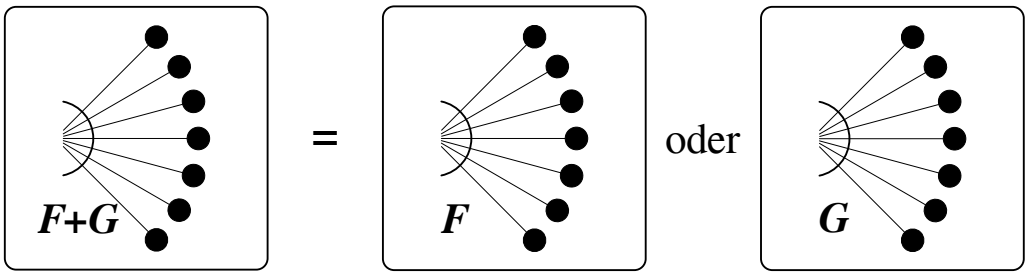

Abbildung 4: Summe von Spezies

2.4 Operationen auf Spezies

In diesem Abschnitt beschäftigen wir uns mit Operationen auf Spezies. Eine Operation erzeugt dabei aus gegebenen Spezies nach gewissen Regeln eine neue Spezies. Im Rahmen dieser Arbeit werden nur jene Operationen betrachtet, die für das Boltzmann-Sampling relevant sind. Dazu gehört die Summenbildung, das Produkt von Spezies, die Substitution, die Ableitung und der Pointing-Operator.

2.4.1 Summe von Spezies

Die erste Operation, die wir betrachten, ist die *Summe von Spezies*. Wenn wir zwei Spezies $\mathcal{F}$ und $\mathcal{G}$ betrachten, so bildet die Summe $\mathcal{F}+\mathcal{G}$ die disjunkte Vereinigung der von $\mathcal{F}$ und $\mathcal{G}$ erzeugten Strukturen.

Definition 2.4.1. Seien $\mathcal{F}$ und $\mathcal{G}$ zwei kombinatorische Spezies. Die Spezies $\mathcal{F}+\mathcal{G}$, auch als *Summe von $\mathcal{F}$ und $\mathcal{G}$* bezeichnet, bildet auf einer beliebigen endlichen Menge U die Strukturen

$$(\mathcal{F}+\mathcal{G})[U] = (\mathcal{F}[U]\times\{1\}) \cup (\mathcal{G}[U]\times\{2\}).$$

Der Transport der Strukturen ist für endliche Mengen U und V und eine Bijektion $\sigma : U \to V$ als

$$(\mathcal{F}+\mathcal{G})[\sigma]((s,i)) = \begin{cases} (\mathcal{F}[\sigma](s),1) & \text{für } i=1,\\ (\mathcal{G}[\sigma](s),2) & \text{für } i=2,\end{cases}$$

definiert.

Proposition 2.4.2. *Seien $\mathcal{F}$ und $\mathcal{G}$ zwei kombinatorische Spezies, dann gilt für die zugehörigen erzeugenden Funktionen*

$$(F+G)(z) = F(z)+G(z) \tag{15}$$

$$\widetilde{(F+G)}(z) = \widetilde{F}(z)+\widetilde{G}(z) \tag{16}$$

$$Z_{F+G}(z_1,z_2,\ldots) = Z_F(z_1,z_2,\ldots)+Z_G(z_1,z_2,\ldots). \tag{17}$$

Beweis. Wir zeigen die Identität für die Zyklenindex-Reihen. Die beiden anderen Identitäten folgen dann aus Satz 2.3.17. Zuerst betrachten wir zu einer gegebenen Permutation $\sigma : [n] \to [n]$ die Anzahl der Fixpunkte fix $(\mathcal{F}+\mathcal{G})[\sigma]$. Es gilt

$$\begin{aligned}
\text{fix } (\mathcal{F}+\mathcal{G})[\sigma] &= |\{s\in(\mathcal{F}+\mathcal{G})[n] \,:\, \mathcal{F}+\mathcal{G}[\sigma](s)=s\}|\\
&= |\{(s,1)\in(\mathcal{F}+\mathcal{G})[n] \,:\, (\mathcal{F}+\mathcal{G})[\sigma]((s,1))=(s,1)\}| +\\
&\quad |\{(s,2)\in(\mathcal{F}+\mathcal{G})[n] \,:\, (\mathcal{F}+\mathcal{G})[\sigma]((s,2))=(s,2)\}|\\
&= |\{(s,1)\in(\mathcal{F}+\mathcal{G})[n] \,:\, (\mathcal{F}[\sigma](s),1)=(s,1)\}| +\\
&\quad |\{(s,2)\in(\mathcal{F}+\mathcal{G})[n] \,:\, (\mathcal{G}[\sigma](s),2)=(s,2)\}|\\
&= |\{s\in\mathcal{F}[n] \,:\, \mathcal{F}[\sigma](s)=s\}| + |\{s\in\mathcal{G}[n] \,:\, \mathcal{G}[\sigma](s)=s\}|\\
&= \text{fix } \mathcal{F}[\sigma] + \text{fix } \mathcal{G}[\sigma].
\end{aligned}$$

Also folgt für die Zyklenindex-Reihen

$$\begin{aligned} Z_{F+G}(z_1, z_2, \ldots) &= \sum_{n\geq 0} \frac{1}{n!} \sum_{\sigma\in\mathcal{P}[n]} \text{fix } (\mathcal{F}+\mathcal{G})\,[\sigma] z_1^{\sigma_1} z_2^{\sigma_2} \cdots \\ &= \sum_{n\geq 0} \frac{1}{n!} \sum_{\sigma\in\mathcal{P}[n]} (\text{fix } \mathcal{F}[\sigma] + \text{fix } \mathcal{G}[\sigma])\, z_1^{\sigma_1} z_2^{\sigma_2} \cdots \\ &= \sum_{n\geq 0} \frac{1}{n!} \sum_{\sigma\in\mathcal{P}[n]} \text{fix } \mathcal{F}[\sigma] z_1^{\sigma_1} z_2^{\sigma_2} \cdots \\ &+ \sum_{n\geq 0} \frac{1}{n!} \sum_{\sigma\in\mathcal{P}[n]} \text{fix } \mathcal{G}[\sigma] z_1^{\sigma_1} z_2^{\sigma_2} \cdots \\ &= Z_F(z_1, z_2, \ldots) \cdot Z_G(z_1, z_2, \ldots). \end{aligned}$$

□

Es ist möglich, die Summen-Operation auf sogenannte *summierbare* Familien von Spezies zu erweitern. Dazu gehen wir folgendermaßen vor:

Definition 2.4.3. Eine Familie $(\mathcal{F}_i)_{i\in I}$ von kombinatorischen Spezies heißt *summierbar* genau dann, wenn für jede endliche Menge U und alle bis auf endlich viele $i \in I$ gilt, dass $\mathcal{F}_i[U] = \emptyset$. Die von der Summenspezies $\sum_{i\in I} \mathcal{F}_i$ einer summierbaren Familie $(\mathcal{F}_i)_{i\in I}$ erzeugten Strukturen sind durch die Gleichung

$$\left(\sum_{i\in I} \mathcal{F}_i\right)[U] = \sum_{i\in I} \mathcal{F}_i[U] = \bigcup_{i\in I} \mathcal{F}_i[U] \times \{i\} \text{ für alle endlichen Mengen } U \tag{18}$$

gegeben. Der Transport der Strukturen ist für endliche Mengen U, V, eine Bijektion $\sigma : U \to V$ und eine beliebige Struktur $(s, i) \in \left(\sum_{i\in I} \mathcal{F}_i\right)[U]$ definiert als

$$\left(\sum_{i\in I} \mathcal{F}_i\right)[\sigma]\,((s, i)) := (\mathcal{F}_i[\sigma](s), i)\,. \tag{19}$$

Dass es sich bei dieser Definition um eine Spezies handelt, folgt daraus, dass für jede endliche Menge U nur endlich viele $\mathcal{F}_i$ einen Beitrag zur Menge der erzeugten Strukturen leisten, und daher nur endlich viele Strukturen auf U erzeugt werden. Genauso wie bei der Summe von zwei Spezies gilt für die erzeugenden Funktionen, dass sich die Summenoperation in die Summe von erzeugenden Funktionen übersetzt.

Proposition 2.4.4. *Sei $(\mathcal{F}_i)_{i\in I}$ eine summierbare Familie von Spezies, dann gilt für die erzeugenden Funktionen*

$$\left(\sum_{i\in I} F_i\right)(z) = \sum_{i\in I} F_i(z), \tag{20}$$

$$\widetilde{\left(\sum_{i\in I} F_i\right)}(z) = \sum_{i\in I} \widetilde{F}_i(z), \tag{21}$$

$$Z_{\left(\sum_{i\in I} F_i\right)}(z_1, z_2, \ldots) = \sum_{i\in I} Z_{F_i}(z_1, z_2, \ldots). \tag{22}$$

Der Beweis verläuft ähnlich wie in Proposition 2.4.2. Die Identität für die Zyklenindex-Reihen gilt, weil es sich bei $(Z_{F_i}(z_1, z_2, \ldots))_{i\in I}$ um eine summierbare Familie von formalen Potenzreihen handelt, da zu jedem Monom $z_1^{n_1} z_2^{n_2} \cdots z_k^{n_k}$ nur endlich viele $i \in I$ existieren, für die die zugehörige Spezies Strukturen auf der Menge $[n]$ erzeugt. Daher können auch nur diese Spezies zur Anzahl der Fixpunkte der Transporte der Strukturen beitragen.

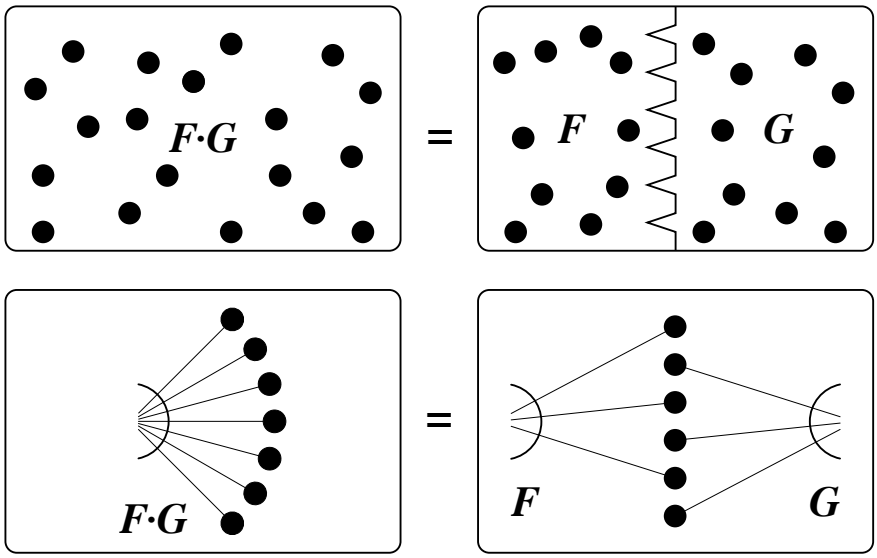

Abbildung 5: Produkt von Spezies

2.4.2 Produkt von Spezies

Nun wenden wir uns dem *Produkt* von Spezies zu. Eine Struktur auf einer endlichen Menge U des Produkts der Spezies $\mathcal{F}$ und $\mathcal{G}$ entsteht dadurch, dass die Menge U in zwei Mengen U_1 und U_2 aufgeteilt wird, d.h. $U_1 \cap U_2 = \emptyset$ und $U_1 \cup U_2 = U$. Daraufhin wird eine $\mathcal{F}$-Struktur s auf U_1 und eine $\mathcal{G}$-Struktur t auf U_2 gebildet und zum Paar (s,t) zusammengefasst. Dies wird in der folgenden Definition formalisiert.

Definition 2.4.5. Seien $\mathcal{F}$ und $\mathcal{G}$ zwei Spezies, dann gilt für die als *Produkt von $\mathcal{F}$ und $\mathcal{G}$* bezeichnete Spezies $\mathcal{F} \cdot \mathcal{G}$ für alle endlichen Mengen U:

$$(\mathcal{F} \cdot \mathcal{G})[U] = \sum_{\substack{(U_1,U_2) \\ U_1 \cap U_2 = \emptyset \\ U_1 \cup U_2 = U}} \mathcal{F}[U_1] \times \mathcal{G}[U_2]. \tag{23}$$

Der Transport der Strukturen ist für endliche Mengen U, V und eine Bijektion $\sigma : U \to V$ für alle $(s,t,U_1,U_2) \in (\mathcal{F} \cdot \mathcal{G})[U]$ definiert als

$$(\mathcal{F} \cdot \mathcal{G})[\sigma]((s,t,U_1,U_2)) := \left(\mathcal{F}[\sigma\big|_{U_1}](s), \mathcal{G}[\sigma\big|_{U_2}](t), \sigma(U_1), \sigma(U_2)\right). \tag{24}$$

Proposition 2.4.6. *Seien $\mathcal{F}$ und $\mathcal{G}$ zwei Spezies, dann erfüllen die zugehörigen erzeugenden Funktionen die Gleichungen*

$$(F \cdot G)(z) = F(z) \cdot G(z) \tag{25}$$

$$\widetilde{(F \cdot G)}(z) = \widetilde{F}(z) \cdot \widetilde{G}(z) \tag{26}$$

$$Z_{(F \cdot G)}(z_1, z_2, \ldots) = Z_F(z_1, z_2, \ldots) \cdot Z_G(z_1, z_2, \ldots), \tag{27}$$

wobei $(F \cdot G)(z)$, $\widetilde{(F \cdot G)}(z)$ und $Z_{(F \cdot G)}(z_1, z_2, \ldots)$ die erzeugenden Funktionen der Spezies $(\mathcal{F} \cdot \mathcal{G})$ sind.

Beweis. Wir zeigen die Identität für die Zyklenindex-Reihen. Die restlichen Identitäten folgen aus Satz 2.3.17.

Betrachten wir zunächst eine beliebige Permutation $\sigma : [n] \to [n]$ mit Zyklentyp $(n_1, \ldots, n_k)$, $\sum_{i=1}^{k} in_i = n$. Eine Struktur $(s, t, U_1, U_2) \in (\mathcal{F} \cdot \mathcal{G})[n]$ ist genau dann ein Fixpunkt von $(\mathcal{F} \cdot \mathcal{G})[\sigma]$, wenn $\sigma(U_1) = U_1$, $\sigma(U_2) = U_2$, $\mathcal{F}[\sigma\big|_{U_1}](s) = s$ und $\mathcal{G}[\sigma\big|_{U_2}](t) = t$. Die Bedingung der Invarianz der Mengen U_1 und U_2 unter der Abbildung σ ist genau dann erfüllt, wenn es keinen Zyklus von σ gibt, der sowohl Elemente von U_1, als auch Elemente von U_2 enthält. Die Zyklen müssen also auf die beiden Mengen aufgeteilt werden. Halten wir nun derartige Mengen U_1 und U_2 fest. Um einen Fixpunkt von $(\mathcal{F} \cdot \mathcal{G})[\sigma]$ zu erhalten, können wir unabhängig voneinander einen Fixpunkt von $\mathcal{F}[\sigma\big|_{U_1}]$ und einen Fixpunkt von $\mathcal{G}[\sigma\big|_{U_2}]$ wählen. Für den Zyklentyp $(k_1, \ldots, k_k)$ von $\mathcal{F}[\sigma\big|_{U_1}]$ und den Zyklentyp $(l_1, \ldots, l_k)$ von $\mathcal{G}[\sigma\big|_{U_2}]$ gilt, dass $k_1 + l_1 = n_1, \ldots, k_k + l_k = n_k$. Außerdem hängt die Anzahl der Fixpunkte von $(\mathcal{F} \cdot \mathcal{G})[\sigma]$, $\mathcal{F}[\sigma\big|_{U_1}]$ und $\mathcal{G}[\sigma\big|_{U_2}]$ nur vom jeweiligen Zyklentyp und nicht von der konkreten Abbildung ab. Es gilt daher

$$\operatorname{fix}(\mathcal{F} \cdot \mathcal{G})[n_1, \ldots, n_k] = \sum_{i_1=0}^{n_1} \cdots \sum_{i_k=0}^{n_k} \binom{n_1}{i_1} \cdots \binom{n_k}{i_k} \cdot \operatorname{fix} \mathcal{F}[i_1, \ldots, i_k] \cdot \operatorname{fix} \mathcal{G}[n_1 - i_1, \ldots, n_k - i_k].$$

Nun wenden wir uns den Zyklenindex-Reihen zu. Es gilt

$$\begin{aligned}
Z_{(\mathcal{F}\cdot\mathcal{G})}(z_1, z_2, \ldots) &= \sum_{n\geq 0} \sum_{n_1+2n_2+\ldots+kn_k=n} \frac{\operatorname{fix}(\mathcal{F} \cdot \mathcal{G})[n_1, \ldots, n_k]}{\operatorname{aut}(n_1, \ldots, n_k)} z_1^{n_1} \cdots z_k^{n_k} \\
&= \sum_{n\geq 0} \sum_{n_1+2n_2+\ldots+kn_k=n} \sum_{i_1=0}^{n_1} \cdots \sum_{i_k=0}^{n_k} \binom{n_1}{i_1} \cdots \binom{n_k}{i_k} \cdot \frac{\operatorname{fix} \mathcal{F}[i_1, \ldots, i_k] \cdot \operatorname{fix} \mathcal{G}[n_1 - i_1, \ldots, n_k - i_k]}{\operatorname{aut}(n_1, \ldots, n_k)} z_1^{n_1} \cdots z_k^{n_k} \\
&= \sum_{n\geq 0} \sum_{\substack{(i_1+j_1)+2(i_2+j_2)+ \\ 3(i_3+j_3)+\ldots+k(i_k+j_k)=n}} \binom{i_1+j_1}{i_1} \cdots \binom{i_k+j_k}{i_k} \cdot \frac{\operatorname{fix} \mathcal{F}[i_1, \ldots, i_k] \cdot \operatorname{fix} \mathcal{G}[j_1, \ldots, j_k]}{\operatorname{aut}(i_1 + j_1, \ldots, i_k + j_k)} z_1^{(i_1+j_1)} \cdots z_k^{(i_k+j_k)} \\
&= \sum_{n\geq 0} \sum_{\substack{(i_1+j_1)+2(i_2+j_2)+ \\ 3(i_3+j_3)+\ldots+k(i_k+j_k)=n}} \left(\frac{\operatorname{fix} \mathcal{F}[i_1, \ldots, i_k]}{\operatorname{aut}(i_1, \ldots, i_k)} z_1^{i_1} \cdots z_k^{i_k} \right) \cdot \left(\frac{\operatorname{fix} \mathcal{G}[j_1, \ldots, j_k]}{\operatorname{aut}(j_1, \ldots, j_k)} z_1^{j_1} \cdots z_k^{j_k} \right) \\
&= \sum_{n\geq 0} \sum_{m=0}^{n} \left(\sum_{i_1+2i_2+\ldots+ki_k=m} \frac{\operatorname{fix} \mathcal{F}[i_1, \ldots, i_k]}{\operatorname{aut}(i_1, \ldots, i_k)} z_1^{i_1} \cdots z_k^{i_k} \right) \cdot \left(\sum_{j_1+2j_2+\ldots+kj_k=n-m} \frac{\operatorname{fix} \mathcal{G}[j_1, \ldots, j_k]}{\operatorname{aut}(j_1, \ldots, j_k)} z_1^{j_1} \cdots z_k^{j_k} \right) \\
&= \left(\sum_{n\geq 0} \sum_{n_1+2n_2+\ldots kn_k=n} \frac{\operatorname{fix} \mathcal{F}[n_1, \ldots, n_k]}{\operatorname{aut}(n_1, \ldots, n_k)} z_1^{n_1} \cdots z_k^{n_k} \right) \cdot \left(\sum_{n\geq 0} \sum_{n_1+2n_2+\ldots kn_k=n} \frac{\operatorname{fix} \mathcal{G}[n_1, \ldots, n_k]}{\operatorname{aut}(n_1, \ldots, n_k)} z_1^{n_1} \cdots z_k^{n_k} \right) \\
&= Z_{\mathcal{F}}(z_1, z_2, \ldots) \cdot Z_{\mathcal{G}}(z_1, z_2, \ldots),
\end{aligned}$$

wobei zu beachten ist, dass

$$\begin{aligned}
\frac{\operatorname{aut}(i_1 + j_1, \ldots, i_k + j_k)}{\operatorname{aut}(i_1, \ldots, i_k) \cdot \operatorname{aut}(j_1, \ldots, j_k)} &= \frac{1^{i_1+j_1} \cdot (i_1 + j_1)! \cdots k^{i_k+j_k} \cdot (i_k + j_k)!}{(1^{i_1} \cdot i_1! \cdots k^{i_k} \cdot i_k!) \cdot (1^{j_1} \cdot j_1! \cdots k^{j_k} \cdot j_k!)} \\
&= \frac{(i_1 + j_1)! \cdots (i_k + j_k)!}{i_1! \cdot j_1! \cdots i_k! \cdot j_k!} = \binom{i_1 + j_1}{i_1} \cdots \binom{i_k + j_k}{i_k}.
\end{aligned}$$

□

2.4.3 Substitution

Die nächste Operation, die wir betrachten, ist die sogenannte *Substitution*. Hierbei wird eine Spezies $\mathcal{G}$ in eine Spezies $\mathcal{F}$ *eingesetzt*. Das bedeutet, dass eine endliche Menge U, auf der eine Struktur erzeugt wird, zuerst einmal partitioniert wird. Dann wird aus der Partition eine $\mathcal{F}$-Struktur gebildet. Schließlich wird auf jeder in der Partition enthaltenen Klasse eine $\mathcal{G}$-Struktur erzeugt. Wir denken uns dann in der $\mathcal{F}$-Struktur die Klassen der Partition durch die korrespondierenden $\mathcal{G}$-Strukturen ersetzt.

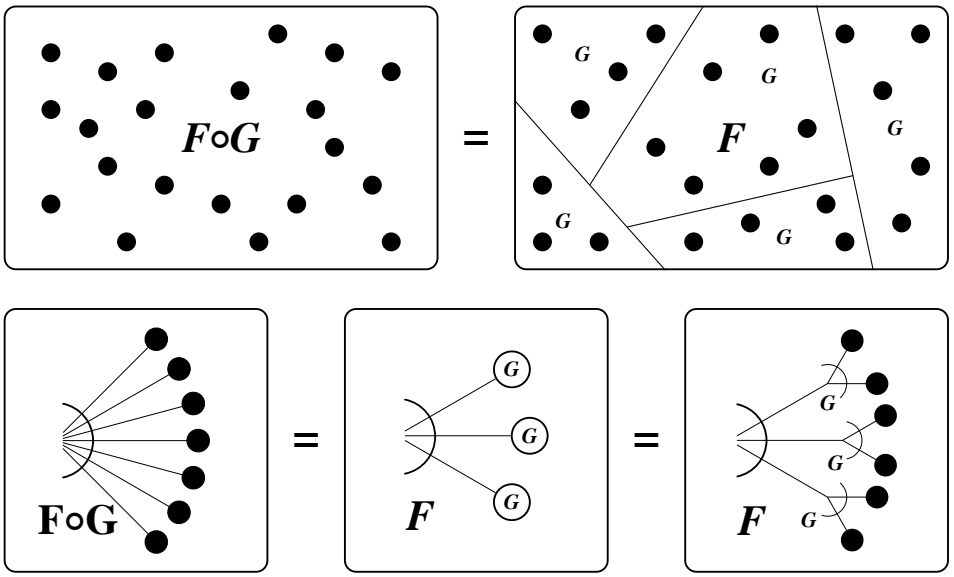

Abbildung 6: Substitution von Spezies

Definition 2.4.7. Seien $\mathcal{F}$ und $\mathcal{G}$ zwei Spezies, sodass keine $\mathcal{G}$-Strukturen auf der leeren Menge existieren, also $\mathcal{G}[\emptyset] = \emptyset$ gilt. Die Spezies $(\mathcal{F} \circ \mathcal{G})$, die wir auch als $\mathcal{F}(\mathcal{G})$ schreiben, und die wir *(partitionelle) Komposition von $\mathcal{F}$ mit $\mathcal{G}$* nennen, ist folgendermaßen definiert: Sei U eine endliche Menge, dann ist eine $(\mathcal{F} \circ \mathcal{G})$-Struktur auf U ein Tripel (π, s, t), wobei

1. π eine Partition von U,
2. s eine $\mathcal{F}$-Struktur auf π, und
3. $t = (t_p)_{p \in \pi}$ ist, wobei t_p eine $\mathcal{G}$-Struktur auf p ist.

Es gilt also

$$(\mathcal{F} \circ \mathcal{G})[U] = \sum_{\pi \in \mathrm{PAR}[U]} \mathcal{F}[\pi] \times \prod_{p \in \pi} \mathcal{G}[p]. \tag{28}$$

Seien nun U und V endliche Mengen und $\sigma : U \to V$ eine Bijektion, dann ist der Transport der $(\mathcal{F} \circ \mathcal{G})$-Strukturen entlang σ als

$$(\mathcal{F} \circ \mathcal{G})[\sigma]\left(\pi, s, (t_p)_{p \in \pi}\right) := \left(\overline{\pi}, \overline{s}, \overline{t}\right) \tag{29}$$

definiert. Hierbei gilt $\overline{\pi} = \mathrm{PAR}[\sigma](\pi)$. Wenn $\pi = \{p_1, \ldots, p_k\}$, dann hat die durch den Transport entstandene Partition $\overline{\pi}$ also die Form $\{\underbrace{\sigma(p_1)}_{\overline{p_1}}, \ldots, \underbrace{\sigma(p_k)}_{\overline{p_k}}\}$. Die Bijektion σ induziert eine Familie von Bijektionen $(\sigma_\pi)_{\pi \in \mathrm{PAR}[U]}$ mit

$$\sigma_\pi : \pi \to \mathrm{PAR}[\sigma](\pi), p \mapsto \sigma(p).$$

Die $\mathcal{F}$-Struktur $\overline{s}$ entsteht nun durch den Transport von s entlang σ_π. Es gilt also $\overline{s} = \mathcal{F}[\sigma_\pi](s)$. Schließlich gilt noch $\overline{t} = \left(\overline{t}_{\overline{p}}\right)_{\overline{p} \in \overline{\pi}}$ mit $\overline{t}_{\overline{p}} = \mathcal{G}[\sigma\big|_p](t_p)$ für $p = \sigma^{-1}(\overline{p})$.

Um ein klareres Bild davon zu bekommen, sehen wir uns ein Beispiel an.

Beispiel 2.4.8. Betrachten wir die Mengen $U := \{1, 2, \ldots, 7\}$, $V := \{a, b, \ldots, g\}$ und die Spezies $\mathrm{SEQ} \circ \mathrm{CYC}$, also endliche Folgen von Zyklen. Eine Struktur dieser Spezies ist intuitiv zum Beispiel

die Folge $((2,4),(5,3),(7,6,1))$. Diese konkrete Struktur wird in $(\textsc{Seq} \circ \textsc{Cyc})[U]$ durch das Tripel (π, s, t) repräsentiert, wobei $\pi = \{\{1,6,7\},\{2,4\},\{3,5\}\}$, $s = (\{2,4\},\{3,5\},\{1,6,7\})$ und $t = \left\{\left(\{1,6,7\}, t_{\{1,6,7\}}\right), \left(\{2,4\}, t_{\{2,4\}}\right), \left(\{3,5\}, t_{\{3,5\}}\right)\right\}$. Es gilt dabei $t_{\{1,6,7\}} = (7,6,1)$, $t_{\{2,4\}} = (2,4)$ und $t_{\{3,5\}} = (5,3)$.

Nun betrachten wir zur Bijektion $\sigma : U \to V$ mit $\sigma(1) = a, \sigma(2) = b, \ldots, \sigma(7) = g$ den Transport der Struktur (π, s, t) entlang σ. Sei $\left(\overline{\pi}, \overline{s}, \overline{t}\right) := (\textsc{Seq} \circ \textsc{Cyc})[\sigma](\pi, s, t)$, dann gilt

$$\overline{\pi} = \textsc{Par}[\sigma](\pi) = \textsc{Par}[\sigma](\{\{1,6,7\},\{2,4\},\{3,5\}\}) = \{\{a,f,g\},\{b,d\},\{c,e\}\}.$$

Die durch σ induzierte Bijektion $\sigma_\pi : \pi \to \overline{\pi}$ bildet $\{1,6,7\}$ auf $\{a,f,g\}$, $\{2,4\}$ auf $\{b,d\}$ und $\{3,5\}$ auf $\{c,e\}$ ab. Wir sehen also, dass

$$\overline{s} = \textsc{Seq}[\sigma_\pi](s) = (\sigma_\pi(\{2,4\}), \sigma_\pi(\{3,5\}), \sigma_\pi(\{1,6,7\})) = (\{b,d\},\{c,e\},\{a,f,g\}).$$

Betrachten wir nun noch die Familie $\overline{t} = \left(\overline{t}_{\overline{p}}\right)_{\overline{p} \in \overline{\pi}}$, so bemerken wir, dass

$$\begin{aligned}
\overline{t}_{\{a,f,g\}} &= \textsc{Cyc}[\sigma\big|_{\{1,6,7\}}](t_{\{1,6,7\}}) = \textsc{Cyc}[\sigma\big|_{\{1,6,7\}}]((7,6,1)) = (g,f,a),\\
\overline{t}_{\{b,d\}} &= \textsc{Cyc}[\sigma\big|_{\{2,4\}}](t_{\{2,4\}}) = \textsc{Cyc}[\sigma\big|_{\{2,4\}}]((2,4)) = (b,d), \text{ und}\\
\overline{t}_{\{c,e\}} &= \textsc{Cyc}[\sigma\big|_{\{3,5\}}](t_{\{3,5\}}) = \textsc{Cyc}[\sigma\big|_{\{3,5\}}]((5,3)) = (e,c).
\end{aligned}$$

Der Transport der Struktur $((2,4),(5,3),(7,6,1))$ entlang σ produziert also wie erwartet die Struktur $((b,d),(e,c),(g,f,a))$.

Bevor wir die zur Operation gehörenden erzeugenden Funktionen betrachten können, müssen wir noch eine Operation auf formalen Potenzreihen definieren.

Definition 2.4.9. Seien $f = f(z_1, z_2, \ldots)$ und $g = g(z_1, z_2, \ldots)$ formale Potenzreihen in unendlich vielen Veränderlichen aus $R[[z_1, z_2, \ldots]]$, dann ist die *plethystische Substitution* $(f \circ g)$ als

$$(f \circ g)(z_1, z_2, \ldots) := f(g_1, g_2, \ldots) \tag{30}$$

definiert, wobei gilt, dass

$$g_k := g(z_k, z_{2k}, z_{3k}, \ldots) \text{ für alle } k \in \mathbb{N}. \tag{31}$$

Satz 2.4.10. *Seien $\mathcal{F}$ und $\mathcal{G}$ zwei Spezies mit $\mathcal{G}[\emptyset] = \emptyset$, dann erfüllen die zugehörigen erzeugenden Funktionen die Gleichungen*

$$(F \circ G)(z) = F(G(z)), \tag{32}$$

$$\widetilde{(F \circ G)}(z) = Z_F(\widetilde{G}(z), \widetilde{G}(z^2), \widetilde{G}(z^3), \ldots), \tag{33}$$

$$Z_{(F \circ G)}(z_1, z_2, z_3, \ldots) = Z_F(Z_G(z_1, z_2, \ldots), Z_G(z_2, z_4, \ldots), \ldots) = (Z_F \circ Z_G)(z_1, z_2, \ldots), \tag{34}$$

wobei $(F \circ G)(z)$, $\widetilde{(F \circ G)}(z)$ und $Z_{(F \circ G)}(z_1, z_2, \ldots)$ die erzeugenden Funktionen der Spezies $(\mathcal{F} \circ \mathcal{G})$ bezeichnen.

Beweis. Der Beweis der Gleichung der Zyklenindexreihen würde den Rahmen dieser Arbeit sprengen und ist ausführlich in [2, Kapitel 4.3, S. 309-315] zu finden. Für die exponentiell erzeugenden Funktionen gilt durch Anwendung von Satz 2.3.17

$$\begin{aligned}
(F \circ G)(z) &= Z_{(F \circ G)}(z, 0, 0, \ldots)\\
&= Z_F(Z_G(z, 0, 0, \ldots), Z_G(0, 0, 0, \ldots), Z_G(0, 0, 0, \ldots), \ldots)\\
&= Z_F(G(z), 0, 0, \ldots) = F(G(z)).
\end{aligned}$$

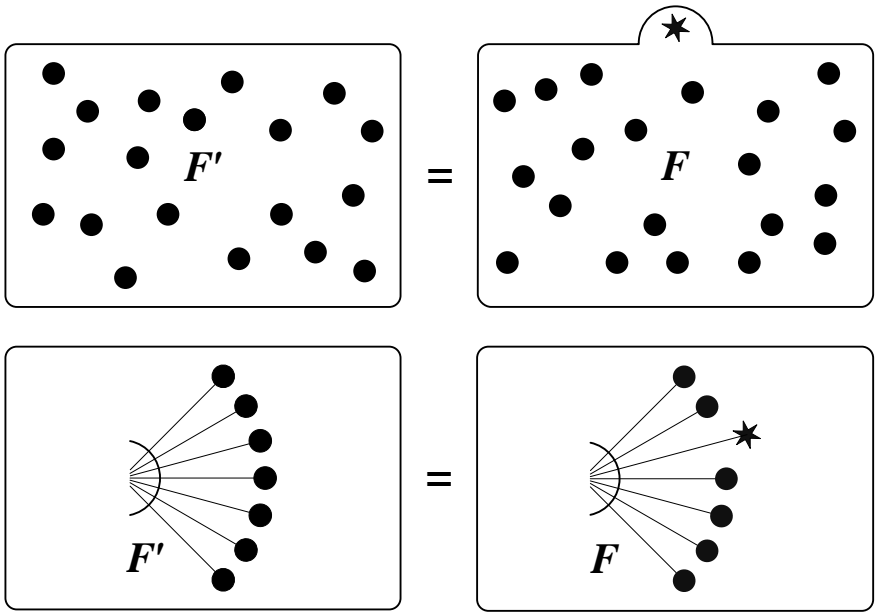

Abbildung 7: Ableitung einer Spezies

Betrachten wir die gewöhnlichen erzeugenden Funktionen, so sehen wir wieder mittels Anwendung von Satz 2.3.17, dass

$$\begin{aligned}(\widetilde{F \circ G})(z) &= Z_{(F\circ G)}(z, z^2, z^3, \ldots)\\ &= Z_F(Z_G(z, z^2, z^3, \ldots), Z_G(z^2, z^4, z^6, \ldots), Z_G(z^3, z^6, z^9, \ldots), \ldots)\\ &= Z_F(Z_G(z, z^2, z^3, \ldots), Z_G((z^2), (z^2)^2, (z^2)^3, \ldots), Z_G((z^3), (z^3)^2, (z^3)^3, \ldots), \ldots)\\ &= Z_F(\widetilde{G}(z), \widetilde{G}(z^2), \widetilde{G}(z^3), \ldots)\end{aligned}$$

gilt. □

Einer der Gründe für die Betrachtung der Zyklenindexreihe ist, dass für die gewöhnlichen erzeugenden Funktionen von zwei Spezies $\mathcal{F}$ und $\mathcal{G}$ im Allgemeinen $(\widetilde{F \circ G})(z) \neq \widetilde{F}(\widetilde{G}(z))$ gilt.

2.4.4 Ableitung

Die Strukturen, die von der Ableitung einer Spezies $\mathcal{F}$ auf einer endlichen Menge U erzeugt werden, entstehen dadurch, dass der Menge U ein nicht in ihr enthaltenes Element hinzugefügt wird, und auf dieser erweiterten Menge dann die $\mathcal{F}$-Strukturen gebildet werden. Diese Operation ist so definiert, dass für die erzeugenden Funktionen der Ableitung einer Spezies gilt, dass sie die Ableitung der erzeugenden Funktionen der Spezies sind.

Definition 2.4.11. Sei $\mathcal{F}$ eine Spezies, dann ist die *Ableitung* der Spezies $\mathcal{F}$, auch als $\mathcal{F}'$ geschrieben, folgendermaßen definiert: Für jede endliche Menge U gilt

$$\mathcal{F}'[U] := \mathcal{F}[U^+] \text{ mit } U^+ := U \cup \{*_U\}\,. \tag{35}$$

Das Element $*_U$ ist hierbei nicht in der Menge U enthalten. Sei nun V eine weitere endliche Menge und $\sigma : U \to V$ eine Bijektion, dann arbeitet der Transport der Strukturen mittels der Vorschrift

$$\mathcal{F}'[\sigma](s) := \mathcal{F}[\sigma^+](s), \tag{36}$$

wobei

$$\sigma^+ : U \cup \{*_U\} \to V \cup \{*_V\}, s \mapsto \begin{cases} \sigma(s) & \text{für } s \in U, \\ *_V & \text{für } s = *_U. \end{cases} \tag{37}$$

Um bei gegebener Menge U ein Element ausserhalb dieser Menge zu wählen, kann man die Menge U selber als Element wählen, da U sich aufgrund des Regularitätsaxioms der Mengenlehre nicht selber enthalten kann.

Proposition 2.4.12. *Sei $\mathcal{F}$ eine Spezies, dann gelten für die zugehörigen erzeugenden Funktionen die Gleichungen*

$$\begin{aligned} F'(z) &= \frac{d}{dz}F(z), \\ \widetilde{F'}(z) &= \left(\frac{\partial}{\partial z_1}Z_F\right)(z, z^2, z^3, \ldots), \\ Z_{F'}(z_1, z_2, \ldots) &= \left(\frac{\partial}{\partial z_1}Z_F\right)(z_1, z_2, \ldots), \end{aligned}$$

wobei $F'(z)$, $\widetilde{F'}(z)$ und $Z_{F'}(z_1, z_2, \ldots)$ die erzeugenden Funktionen der Spezies $\mathcal{F}'$ bezeichnen.

Beweis. Um die Gleichheit der Zyklenindexreihe zu zeigen, betrachten wir zuerst eine beliebige Permutation $\sigma : [n] \to [n]$. Die Permutation $\sigma^+ : [n+1] \to [n+1]$ besitzt dann genau einen Fixpunkt mehr, nämlich $n+1$. Das Element $n+1$ übernimmt hier die Rolle von $*_{[n]}$. Wenn $(k_1, k_2, \ldots, k_l)$ der Zyklentyp von σ ist, dann ist also $(k_1+1, k_2, \ldots, k_l)$ der Zyklentyp von σ^+. Es gilt also $\operatorname{fix}\mathcal{F}'[k_1, k_2, \ldots, k_l] = \operatorname{fix}\mathcal{F}[k_1+1, k_2, \ldots, k_l]$. Nun gilt

$$\begin{aligned} \left(\frac{\partial}{\partial z_1}Z_F\right)(z_1, z_2, \ldots) &= \frac{\partial}{\partial z_1}\sum_{n\geq 0}\sum_{k_1+2k_2+\ldots+lk_l} \frac{\operatorname{fix}\mathcal{F}[k_1, k_2, \ldots, k_l]}{\operatorname{aut}(k_1, k_2, \ldots, k_l)} z_1^{k_1} z_2^{k_2} \cdots z_l^{k_l} \\ &= \sum_{n\geq 0}\sum_{k_1+2k_2+\ldots+lk_l=n} k_1 \frac{\operatorname{fix}\mathcal{F}[k_1, k_2, \ldots, k_l]}{\operatorname{aut}(k_1, k_2, \ldots, k_l)} z_1^{k_1-1} z_2^{k_2} \cdots z_l^{k_l} \\ &= \sum_{n\geq 0}\sum_{k_1=1}^{n}\sum_{k_1+2k_2+\ldots lk_l=n} k_1 \frac{\operatorname{fix}\mathcal{F}[k_1, k_2, \ldots, k_l]}{\operatorname{aut}(k_1, k_2, \ldots, k_l)} z_1^{k_1-1} z_2^{k_2} \cdots z_l^{k_l} \\ &= \sum_{n\geq 1}\sum_{k_1=0}^{n-1}\sum_{k_1+2k_2+\ldots+lk_l=n-1} (k_1+1) \frac{\operatorname{fix}\mathcal{F}[k_1+1, k_2, \ldots, k_l]}{\operatorname{aut}(k_1+1, k_2, \ldots, k_l)} z_1^{k_1} z_2^{k_2} \cdots z_l^{k_l} \\ &= \sum_{n\geq 0}\sum_{k_1+2k_2+\ldots+lk_l=n} (k_1+1) \frac{\operatorname{fix}\mathcal{F}[k_1+1, k_2, \ldots, k_l]}{\operatorname{aut}(k_1+1, k_2, \ldots, k_l)} z_1^{k_1} z_2^{k_2} \cdots z_l^{k_l} \\ &= \sum_{n\geq 0}\sum_{k_1+2k_2+\ldots+lk_l=n} \frac{\operatorname{fix}\mathcal{F}[k_1+1, k_2, \ldots, k_l]}{\operatorname{aut}(k_1, k_2, \ldots, k_l)} z_1^{k_1} z_2^{k_2} \cdots z_l^{k_l} \\ &= \sum_{n\geq 0}\sum_{k_1+2k_2+\ldots+lk_l=n} \frac{\operatorname{fix}\mathcal{F}'[k_1, k_2, \ldots, k_l]}{\operatorname{aut}(k_1, k_2, \ldots, k_l)} z_1^{k_1} z_2^{k_2} \cdots z_l^{k_l} = Z_{F'}(z_1, z_2, \ldots), \end{aligned}$$

wenn wir noch berücksichtigen, dass

$$\begin{aligned} \frac{1}{k_1+1} \cdot \operatorname{aut}(k_1+1, k_2, \ldots, k_l) &= \frac{1}{k_1+1} \cdot 1^{k_1+1}(k_1+1)!2^{k_2}k_2! \cdots l^{k_l}k_l! \\ &= 1^{k_1}k_1!2^{k_2}k_2! \cdots l^{k_l}k_l! = \operatorname{aut}(k_1, k_2, \ldots, k_l). \end{aligned}$$

Für die exponentiell erzeugenden Funktionen gilt nun aufgrund von Satz 2.3.17

$$\begin{aligned} F'(z) = Z_{F'}(z, 0, 0, \ldots) &= \sum_{n\geq 0}\sum_{k_1+2k_2+\ldots+lk_l=n} \frac{\operatorname{fix}\mathcal{F}[k_1+1, k_2, \ldots, k_l]}{\operatorname{aut}(k_1, k_2, \ldots, k_l)} z^{k_1} 0^{k_2} \cdots 0^{k_l} \\ &= \sum_{n\geq 0} \frac{\operatorname{fix}\mathcal{F}[n+1, 0, \ldots, 0]}{\operatorname{aut}(n, 0, \ldots, 0)} z^n = \sum_{n\geq 0} \frac{\operatorname{fix}\mathcal{F}[\mathrm{id}_{[n+1]}]}{n!} z^n = \sum_{n\geq 0} \frac{f_{n+1}}{n!} z^n = \sum_{n\geq 1} \frac{f_n}{(n-1)!} z^{n-1} \\ &= \sum_{n\geq 1} n \cdot \frac{f_n}{n!} z^{n-1} = \frac{d}{dz}F(z). \end{aligned}$$

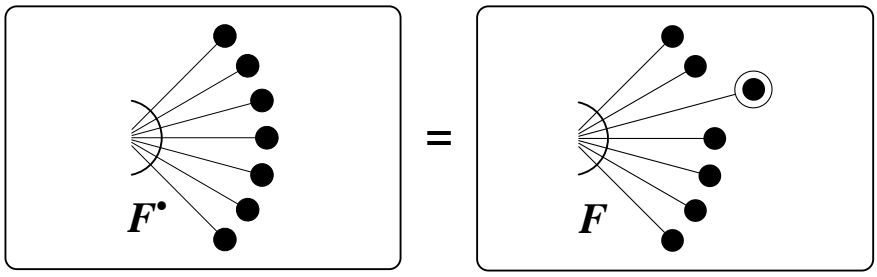

Abbildung 8: Punktierung einer Spezies

Dabei ist zu beachten, dass $z^{k_1}0^{k_2}\cdots 0^{k_l}$ genau dann ungleich 0 ist, wenn $k_1 = n, k_2 = 0, \ldots, k_l = 0$ gilt. Die Identität der gewöhnlichen erzeugenden Funktion ergibt sich direkt aus Satz 2.3.17. □

2.4.5 Punktierung

Eine Operation, die eng mit der Ableitung verwandt ist, ist die *Punktierung*. Eine Struktur der punktierten Spezies $\mathcal{F}$ auf einer endlichen Menge U ist eine $\mathcal{F}$-Struktur, bei der zusätzlich noch ein Element der Grundmenge U ausgezeichnet wird.

Definition 2.4.13. Sei $\mathcal{F}$ eine Spezies, dann wird die Spezies $\mathcal{F}^\bullet$ als *Punktierung* der Spezies $\mathcal{F}$ bezeichnet. Eine $\mathcal{F}^\bullet$-Struktur auf einer endlichen Menge U ist ein Paar (s,t) mit $s \in \mathcal{F}[U]$ und $t \in U$. Es gilt also

$$\mathcal{F}^\bullet[U] = \mathcal{F}[U] \times U. \tag{38}$$

Für eine weitere endliche Menge V und eine Bijektion $\sigma : U \to V$ ist der Transport der Strukturen als

$$\mathcal{F}^\bullet[\sigma](s,t) := (\mathcal{F}[\sigma](s), \sigma(t)) \text{ für alle } (s,t) \in \mathcal{F}^\bullet[U] \tag{39}$$

defniert.

Wir zeigen nun die Verwandtschaft mit der Ableitung. Daraus folgen dann auch leicht die Eigenschaften der erzeugenden Funktionen.

Lemma 2.4.14. *Sei $\mathcal{F}$ eine Spezies, dann gilt $\mathcal{F}^\bullet \simeq \mathcal{Z} \cdot \mathcal{F}'$.*

Beweis. Betrachten wir zuerst die Strukturen der Spezies $\mathcal{Z} \cdot \mathcal{F}'$ auf einer endlichen Menge U. Es gilt

$$\left(\mathcal{Z} \cdot \mathcal{F}'\right)[U] = \sum_{\substack{(U_1,U_2)\\ U_1 \cap U_2 = \emptyset\\ U_1 \cup U_2 = U}} \overbrace{\mathcal{Z}[U_1]}^{\neq\emptyset \Leftrightarrow |U_1|=1} \times \mathcal{F}'[U_2] = \sum_{(\{u\}, U\setminus\{u\}), u\in U} \{u\} \times \mathcal{F}'[U \setminus \{u\}].$$

Eine Struktur aus $\left(\mathcal{Z} \cdot \mathcal{F}'\right)[U]$ hat also die Form $(u, f', \{u\}, U \setminus \{u\})$ mit $u \in U$ und $f' \in \mathcal{F}'[U \setminus \{u\}]$ $= \mathcal{F}[(U \setminus \{u\}) \cup \{*_{U\setminus\{u\}}\}]$. Wenn V eine weitere endliche Menge und $\sigma : U \to V$ eine Bijektion ist, und wir den Transport der Strukturen betrachten, sehen wir, dass

$$\begin{aligned}\left(\mathcal{Z} \cdot \mathcal{F}'\right)[\sigma]\left(u, f', \{u\}, U \setminus \{u\}\right) &= \left(\mathcal{Z}[\sigma|_{\{u\}}](u), \mathcal{F}'[\sigma|_{U\setminus\{u\}}](f'), \sigma(\{u\}), \sigma(U \setminus \{u\})\right)\\ &= \left(\sigma(u), \mathcal{F}\left[\left(\sigma|_{U\setminus\{u\}}\right)^+\right](f'), \{\sigma(u)\}, V \setminus \{\sigma(u)\}\right).\end{aligned}$$

Eine Struktur der Spezies $\mathcal{F}^\bullet[U]$ hat die Gestalt (f,u) mit $f \in \mathcal{F}[U]$ und $u \in U$. Sie wird durch den Transport der Strukturen entlang σ auf die Struktur $(\mathcal{F}[\sigma](f), \sigma(u))$ abgebildet.

Um die behauptete Isomorophie zu zeigen, konstruieren wir nun eine Familie von Bijektionen $(\alpha_U : \mathcal{F}^\bullet[U] \to \left(\mathcal{Z} \cdot \mathcal{F}'\right)[U])$ auf den endlichen Mengen, sodass für alle $s \in \mathcal{F}^\bullet$ gilt, dass

$(\mathcal{Z}\cdot\mathcal{F}')[\sigma](\alpha_U(f,u)) = \alpha_V(\mathcal{F}^\bullet[\sigma](f,u))$. Wir definieren für eine beliebige endliche Menge M und ein Element $m \in M$ die Bijektion

$$\tau_{(M,m)} : M \to (M\setminus\{m\}) \cup \left\{*_{M\setminus\{m\}}\right\}, x \mapsto \begin{cases} x & \text{für } x \in M\setminus\{m\}, \\ *_{M\setminus\{m\}} & \text{für } x = m. \end{cases}$$

Wenn wir die Abbildungen α_U als

$$\alpha_U : \mathcal{F}^\bullet[U] \to \left(\mathcal{Z}\cdot\mathcal{F}'\right)[U], (f,u) \mapsto \left(u, \mathcal{F}[\tau_{(U,u)}](f), \{u\}, U\setminus\{u\}\right)$$

festlegen, dann gilt für alle $(f,u) \in \mathcal{F}^\bullet[U]$:

$$\begin{aligned} & \left(\mathcal{Z}\cdot\mathcal{F}'\right)[\sigma](\alpha_U(f,u)) = \alpha_V(\mathcal{F}^\bullet[\sigma](f,u)) \\ \Leftrightarrow\quad & \left(\mathcal{Z}\cdot\mathcal{F}'\right)[\sigma](u, \mathcal{F}[\tau_{(U,u)}](f), \{u\}, U\setminus\{u\}) = \alpha_V(\mathcal{F}[\sigma](f), \sigma(u)) \\ \Leftrightarrow\quad & \left(\sigma(u), \mathcal{F}\left[\left(\sigma\big|_{U\setminus\{u\}}\right)^+\right](\mathcal{F}[\tau_{(U,u)}](f)), \{\sigma(u)\}, V\setminus\{\sigma(u)\}\right) \\ & = \left(\sigma(u), \mathcal{F}[\tau_{(V,\sigma(u))}](\mathcal{F}[\sigma](f)), \{\sigma(u)\}, V\setminus\{\sigma(u)\}\right) \\ \Leftrightarrow\quad & \mathcal{F}\left[\left(\sigma\big|_{U\setminus\{u\}}\right)^+\right](\mathcal{F}[\tau_{(U,u)}](f)) = \mathcal{F}[\tau_{(V,\sigma(u))}](\mathcal{F}[\sigma](f)) \\ \Leftrightarrow\quad & \mathcal{F}\left[\left(\sigma\big|_{U\setminus\{u\}}\right)^+ \circ \tau_{(U,u)}\right](f) = \mathcal{F}[\tau_{(V,\sigma(u))}\circ\sigma](f). \end{aligned}$$

Es genügt also, zu zeigen, dass $\left(\sigma\big|_{U\setminus\{u\}}\right)^+ \circ \tau_{(U,u)} = \tau_{(V,\sigma(u))}\circ\sigma$. Nun gilt einerseits für ein beliebiges $x \in U$

$$\left(\sigma\big|_{U\setminus\{u\}}\right)^+ \circ \tau_{(U,u)}(x) = \begin{cases} \sigma(\tau_{(U,u)}(x)) & \text{für } \tau_{(U,u)}(x) \in U\setminus\{u\} \\ *_{V\setminus\{\sigma(u)\}} & \text{für } \tau_{(U,u)}(x) = *_{U\setminus\{u\}} \end{cases} = \begin{cases} \sigma(x) & \text{für } x \in U\setminus\{u\} \\ *_{V\setminus\{\sigma(u)\}} & \text{für } x = u, \end{cases}$$

andererseits aber auch

$$\tau_{(V,\sigma(u))}\circ\sigma(x) = \begin{cases} \sigma(x) & \text{für } \sigma(x) \in V\setminus\{\sigma(u)\} \\ *_{V\setminus\{\sigma(u)\}} & \text{für } \sigma(x) = \sigma(u) \end{cases} = \begin{cases} \sigma(x) & \text{für } x \in U\setminus\{u\} \\ *_{V\setminus\{\sigma(u)\}} & \text{für } x = u. \end{cases}$$

Daher sind die Spezies $\mathcal{F}^\bullet$ und $\mathcal{Z}\cdot\mathcal{F}'$ isomorph. □

Proposition 2.4.15. *Sei $\mathcal{F}$ eine Spezies, dann gelten die Gleichungen*

$$F^\bullet(z) = z\cdot\frac{d}{dz}F(z) \tag{40}$$

$$\widetilde{F^\bullet}(z) = z\cdot\left(\frac{\partial}{\partial z_1}Z_F\right)(z, z^2, z^3, \ldots) \tag{41}$$

$$Z_{F^\bullet}(z_1, z_2, \ldots) = z_1\cdot\left(\frac{\partial}{\partial z_1}Z_F\right)(z_1, z_2, \ldots) \tag{42}$$

wobei $\mathcal{F}^\bullet(z)$, $\widetilde{F^\bullet}(z)$ und $Z_{F^\bullet}(z_1, z_2, \ldots)$ die erzeugenden Funktionen der Spezies $F^\bullet$ bezeichnen.

Beweis. Wegen $\mathcal{F}^\bullet \simeq \mathcal{Z}\cdot\mathcal{F}'$ gilt für die erzeugenden Funktionen durch Anwendung der Isomorphie, der Proposition 2.4.6, und der Proposition 2.4.12

$$\begin{aligned} F^\bullet(z) &= (Z\cdot F')(z) = Z(z)\cdot F'(z) = z\cdot\frac{d}{dz}F(z), \\ \widetilde{F^\bullet}(z) &= \widetilde{(Z\cdot F')}(z) = \widetilde{Z}(z)\cdot\widetilde{F'}(z) = z\cdot\left(\frac{\partial}{\partial z_1}Z_F\right)(z, z^2, z^3, \ldots), \text{ und} \\ Z_{F^\bullet}(z_1, z_2, \ldots) &= Z_{Z\cdot F'}(z_1, z_2, \ldots) = Z_Z(z_1, z_2, \ldots)\cdot Z_{F'}(z_1, z_2, \ldots) \\ &= z_1\cdot\left(\frac{\partial}{\partial z_1}Z_F\right)(z_1, z_2, \ldots). \end{aligned}$$

□

2.5 Mehrsortige Spezies

Wir können die Theorie der kombinatorischen Spezies so erweitern, dass Strukturen nicht nur aus einer Menge, sondern aus mehreren Mengen gleichzeitig erzeugt werden. Diese Mengen werden *Sorten*, und eine derartige Spezies wird *mehrsortige Spezies* genannt. Zu vielen Sätzen dieses Abschnitts werden keine Beweise angegeben, da diese Sätze im Prinzip analog zu den bereits in früheren Abschnitten angegebenen Sätzen im einsortigen Fall bewiesen werden und die Beweise aufgrund der komplexen Notation den Rahmen dieser Arbeit sprengen würden. Eine umfangreichere Einführung findet man in [2, S. 100-112].

Bevor wir mehrsortige Spezies behandeln können, müssen wir noch einige Begriffe definieren.

Definition 2.5.1. Sei $k \geq 1$ eine natürliche Zahl. Eine *Multimenge (mit k Sorten von Elementen)* oder kurz k-Menge ist ein k-Tupel von Mengen

$$U = (U_1, \ldots, U_k)\,. \tag{43}$$

Ein Element $u \in U_i$ wird *Element der Sorte i* genannt. Die Multimege $U = (U_1, \ldots, U_k)$ heißt genau dann *endlich*, wenn für alle $i \in \{1, \ldots, k\}$ gilt, dass die Menge U_i endlich ist. Die *Multikardinalität* von U ist das k-Tupel der Kardinalitäten

$$|U| := (|U_1|, \ldots, |U_k|)\,, \tag{44}$$

die totale Kardinalität von U ist die Summe

$$\|U\| := |U_1| + \ldots + |U_k|\,. \tag{45}$$

Ausgezeichnete Mengen, Mengenrelationen und Mengenoperationen können in den Kontext der Multimengen übertragen werden.

Definition 2.5.2. Die *leere k-Menge* ist das k-Tupel $(\underbrace{\emptyset, \ldots, \emptyset}_{k\text{-mal}})$. Seien $U := (U_1, \ldots, U_k)$ und $V := (V_1, \ldots, V_k)$ k-sortige Mengen, dann ist U eine Teilmenge von V, in Zeichen $U \subseteq V$ genau dann, wenn für alle $i \in \{1, \ldots, k\}$ gilt, dass $V_i \subseteq U_i$. Die k-Mengen U und V sind gleich, in Zeichen $U = V$, genau dann, wenn für alle $i \in \{1, \ldots, k\}$ gilt, dass $U_i = V_i$. Sei $(U_i)_{i \in I} := \big((U_{1,i}, \ldots, U_{k,i})\big)_{i \in I}$ eine Familie von k-sortigen Mengen, dann definieren wir

$$\bigcup_{i \in I} U_i := \Big(\bigcup_{i \in I} U_{1,i}, \ldots, \bigcup_{i \in I} U_{k,i}\Big) \text{ und} \tag{46}$$

$$\bigcap_{i \in I} U_i := \Big(\bigcap_{i \in I} U_{1,i}, \ldots, \bigcap_{i \in I} U_{k,i}\Big). \tag{47}$$

Schließlich definieren wir die mengentheoretische Differenz der k-Mengen $U := (U_1, \ldots, U_k)$ und $V := (V_1, \ldots, V_k)$ als

$$U \setminus V := (U_1 \setminus V_1, \ldots, U_k \setminus V_k). \tag{48}$$

Definition 2.5.3. Sei $U = (U_1, U_2, \ldots, U_k)$ eine k-sortige Menge, dann heißt ein Paar (V, W) mit $V = (V_1, V_2, \ldots, V_k)$, $W = (W_1, W_2, \ldots, W_k)$ von k-sortigen Mengen eine *Dissektion* von U genau dann, wenn für alle $i \in \{1, \ldots, k\}$ gilt, dass $V_i \cap W_i = \emptyset$ und $V_i \cup W_i = U_i$. Die Menge der Dissektionen von U wird als $\Delta(U)$ bezeichnet.

Eine *Partition* von U ist eine Menge P von k-Mengen, für die gilt, dass

1. $\emptyset \notin P$,
2. $\bigcup_{M \in P} M = U$, und
3. für alle $M, N \in P, M \neq N$ gilt $M \cap N = \emptyset$.

Die Menge der Partitionen von U wird als $\textsc{Par}[U]$ bezeichnet.

Definition 2.5.4. Seien $U := (U_1, \ldots, U_k)$ und $V := (V_1, \ldots, V_k)$ k-Mengen, dann ist eine *Multifunktion* f von U nach V, auch als

$$f : (U_1, \ldots, U_k) \to (V_1, \ldots, V_k)$$

bezeichnet, ein k-Tupel von Funktionen $f = (f_1, \ldots, f_k)$, wobei für alle $i \in \{1, \ldots, n\}$ gilt, dass $f_i : U_i \to V_i$. f wird genau dann bijektiv genannt, wenn alle f_i bijektiv sind. Sei $W := (W_1, \ldots, W_k)$ eine weitere k-Menge, die Teilmenge von $(U_1, \ldots, U_k)$ ist, dann ist die *Einschränkung der Multifunktion f auf die Multimenge W* als

$$f\big|_W := \left(f_1\big|_{W_1}, \ldots, f_k\big|_{W_k}\right)$$

definiert. Das Bild der k-Menge W unter der Multifunktion f, in Zeichen $f(W)$, ist die k-Menge $(f_1(W_1), \ldots, f_k(W_k))$. Für eine weitere k-Menge $X := (X_1, \ldots, X_n)$ und eine Multifunktion $g : (V_1, \ldots, V_k) \to (X_1, \ldots, X_n)$ ist die Komposition mit f als

$$f \circ g : (U_1, \ldots, U_k) \to (X_1, \ldots, X_n) := (f_1 \circ g_1, \ldots, f_k \circ g_k)$$

definiert.

Definition 2.5.5. Sei $k \geq 1$ eine natürliche Zahl. Eine *Spezies von k Sorten* oder auch *k-sortige Spezies* ist eine Regel $\mathcal{F}$, die

1. für jede endliche k-Menge $U = (U_1, \ldots, U_k)$ eine endliche Menge $\mathcal{F}[U_1, \ldots, U_k]$ erzeugt, und
2. für jede bijektive Multifunktion

   $$\sigma = (\sigma_1, \ldots, \sigma_k) : (U_1, \ldots, U_k) \to (V_1, \ldots, V_k)$$

   auf den endlichen k-Mengen $(U_1, \ldots, U_k)$ und $(V_1, \ldots, V_k)$ eine Funktion

   $$\mathcal{F}[\sigma] = \mathcal{F}[\sigma_1, \ldots, \sigma_k] : \mathcal{F}[U_1, \ldots, U_k] \to \mathcal{F}[V_1, \ldots, V_k]$$

   erzeugt.

Die Funktionen $\mathcal{F}[\sigma]$ müssen dabei folgende Eigenschaften besitzen:

1. Für alle bijektiven Multifunktionen $\sigma : U \to V$ und $\tau : V \to W$ auf den endlichen k-Mengen U, V und W muss gelten, dass

   $$\mathcal{F}[\tau \circ \sigma] = F[\tau] \circ F[\sigma]. \tag{49}$$

2. Für jede endliche Multimenge U muss gelten, dass

   $$\mathcal{F}[\mathrm{id}_U] = \mathrm{id}_{\mathcal{F}[U]}, \tag{50}$$

   wobei id_U die identische Multifunktion $\mathrm{id}_U = (\mathrm{id}_{U_1}, \ldots, \mathrm{id}_{U_k})$ bezeichnet.

Ein Element $s \in \mathcal{F}[U_1, \ldots, U_k]$ wird als *$\mathcal{F}$-Struktur auf $(U_1, \ldots, U_k)$* oder auch als *Struktur der Spezies $\mathcal{F}$ auf $(U_1, \ldots, U_k)$* bezeichnet. Die Funktion $\mathcal{F}[\sigma_1, \ldots, \sigma_k]$ heißt *Transport der $\mathcal{F}$-Strukturen entlang $(\sigma_1, \ldots, \sigma_k)$*.

Die Isomorphie zweier Strukturen und die sich daraus ergebenden Isomorphietypen sind analog zum einsortigen Fall definiert, unter Verwendung von bijektiven Multifunktionen an Stelle von Bijektionen.

Definition 2.5.6. Seien $\mathcal{F}$ eine k-sortige kombinatorische Spezies, U und V k-sortige Mengen und $s_1 \in \mathcal{F}[U]$, $s_2 \in \mathcal{F}[V]$, dann heißt eine bijektive Multifunktion $\sigma : U \to V$ ein *Isomorphismus* zwischen s_1 und s_2 genau dann, wenn $\mathcal{F}[\sigma](s_1) = s_2$. Die Strukturen s_1 und s_2 haben genau in diesem Fall den selben *Isomorphietyp*. Sei $s \in \mathcal{F}[U]$, dann nennen wir einen Isomorphismus zwischen s und s einen *Automorphismus*.

Wie im einsortigen Fall ist die *Isomorphietyprelation* einer mehrsortigen Spezies $\mathcal{F}$, bezüglich der zwei Strukturen auf der selben Grundmultimenge U genau dann in Relation stehen, wenn ein Isomorphismus zwischen ihnen existiert, eine Äquivalenzrelation, deren Äquivalenzklassen wir als *Isomorphietypen* bezeichnen. Die Relation wird auch als $\sim_U$ geschrieben.

Definition 2.5.7. Sei $\mathcal{F}$ eine k-sortige Spezies und $U = (U_1, \ldots, U_k)$ eine k-sortige Menge, dann bezeichnet $T(\mathcal{F}, U)$ die Menge der Isomorphietypen $\mathcal{F}[U]/\sim_U$. Ist $U = ([n_1], \ldots, [n_k])$, dann schreiben wir abkürzend $T(\mathcal{F}, n_1, \ldots, n_k)$ an Stelle von $T(\mathcal{F}, ([n_1], \ldots, [n_k]))$.

Wie im einsortigen Fall können wir auch einer mehrsortigen Spezies erzeugende Funktionen zuordnen. Die exponentiell erzeugende Funktion enthält die Information über die Anzahl der Strukturen in Abhängigkeit der Kardinalität der Grundmenge, die gewöhnliche erzeugende Funktion zählt die Isomorphietypen, und die Zyklenindexreihe enthält hinreichend Information, um die anderen beiden erzeugenden Funktionen daraus abzuleiten.

Definition 2.5.8. Sei $\mathcal{F}$ eine k-sortige Spezies und $(z_i)_{i=1..k}$ eine Familie von Unstimmten, dann ist die *exponentiell erzeugende Funktion von* $\mathcal{F}$, auch als $F(z_1, \ldots, z_k)$ bezeichnet, die formale Potenzreihe

$$F(z_1, \ldots, z_k) = \sum_{i_1, \ldots, i_k \geq 0} f_{i_1, \ldots, i_k} \frac{z_1^{i_1} \cdots z_k^{i_k}}{i_1! \cdots i_k!},$$

wobei $f_{i_1,\ldots,i_k} = \left|\mathcal{F}\left[([i_1], \ldots, [i_k])\right]\right|$ gilt.

Wie im einsortigen Fall ist hier aufgrund der Definition des Transports der Strukturen die Anzahl der Strukturen auf einer Grundmultimenge nur von den Kardinalitäten der einzelnen Sorten und nicht von den konkreten Mengen abhängig.

Definition 2.5.9. Sei $\mathcal{F}$ eine k-sortige Spezies und $(z_i)_{i=1..k}$ eine Familie von Unbestimmten, dann ist die *gewöhnliche erzeugende Funktion von* $\mathcal{F}$, auch als $\widetilde{F}(z_1, \ldots, z_k)$ bezeichnet, die formale Potenzreihe

$$\widetilde{F}(z_1, \ldots, z_k) = \sum_{i_1, \ldots, i_k \geq 0} \tilde{f}_{i_1, \ldots, i_k} z_1^{i_1} \cdots z_k^{i_k},$$

wobei $\tilde{f}_{i_1,\ldots,i_k} = |T(\mathcal{F}, i_1, \ldots, i_k)|$ gilt. Sie wird auch *typerzeugende Funktion der Spezies* $\mathcal{F}$ genannt.

Definition 2.5.10. Sei $\mathcal{F}$ eine k-sortige Spezies und $(z_{i,j})_{\substack{i=1..k \\ j\geq 0}}$ eine Familie von Unbestimmten, dann wird die formale Potenzreihe

$$Z_F(z_{1,1}, z_{1,2}, \ldots; \ldots; z_{k,1}, z_{k,2}, \ldots) = \sum_{i_1, \ldots, i_k \geq 0} \frac{1}{i_1! \cdots i_k!} \sum_{\sigma_1 \in S_{i_1}, \ldots, \sigma_k \in S_{i_k}} \operatorname{fix} \mathcal{F}[\sigma_1, \ldots, \sigma_k] \prod_{l=1}^{k} z_{l,1}^{\sigma_{l,1}} \cdots z_{l,i_l}^{\sigma_{l,i_l}}$$

als *Zyklenindexreihe* der Spezies $\mathcal{F}$ bezeichnet, wobei $(\sigma_{l,1}, \sigma_{l,2}, \ldots, \sigma_{l,i_l})$ den Zyklenindex der Permutation $\sigma_l \in S_{i_l}$ bezeichnet.

Der Beweis der Reduzierung der Zyklenindexreihe auf die exponentiell und die gewöhnliche erzeugende Funktion verläuft analog zum Beweis von Satz 2.3.17. Daher wird der folgende Satz ohne Beweis angegeben.

Satz 2.5.11. *Sei $\mathcal{F}$ eine k-sortige Spezies und $Z_F(z_{1,1}, z_{1,2}, \ldots; \ldots; z_{k_1}, z_{k,2}, \ldots)$ die ihr zugeordnete Zyklenindexreihe, dann gilt für ihre exponentiell erzeugende Funktion $F(z_1, \ldots, z_k)$ und ihre gewöhnliche erzeugende Funktion $\widetilde{F}(z_1, \ldots, z_k)$*

$$F(z_1, \ldots, z_k) = Z_F(z_1, 0, 0, \ldots; \ldots; z_k, 0, 0, \ldots) \text{ und}$$
$$\widetilde{F}(z_1, \ldots, z_k) = \widetilde{F}(z_1, z_1^2, z_1^3, \ldots; \ldots; z_k, z_k^2, z_k^3, \ldots).$$

Nun wenden wir uns speziellen mehrsortigen Spezies und den Operationen, die auf mehrsortigen Spezies operieren, zu.

Definition 2.5.12. Die k-sortige Spezies $\mathcal{Z}_i$ für $1 \leq i \leq k$ erzeugt genau eine Struktur, wenn die i-te Sorte aus genau einem Element besteht, und alle anderen Sorten leer sind. Ansonsten erzeugt sie keine Struktur. Auf einer endlichen k-Menge $U := (U_1, \ldots, U_k)$ leistet $\mathcal{Z}_i$ also

$$\mathcal{Z}_i[U] = \begin{cases} \{U\}, & \text{falls } |U_i| = 1 \text{ und } U_j = \emptyset \text{ für alle } j \neq i, \\ \emptyset & \text{sonst.} \end{cases}$$

Der Transport der Strukturen ist analog zum einsortigen Fall definiert. Die Spezies $\mathcal{Z}_i$ wird auch als *Singleton der i-ten Sorte* bezeichnet.

Definition 2.5.13. Seien $\mathcal{F}$ und $\mathcal{G}$ zwei k-sortige Spezies. Die *Summe von $\mathcal{F}$ und $\mathcal{G}$*, in Zeichen $\mathcal{F} + \mathcal{G}$, bildet auf einer beliebigen endlichen k-Menge U die Strukturen

$$(\mathcal{F} + \mathcal{G})[U] = (\mathcal{F}[U] \times \{1\}) \cup (\mathcal{G}[U] \times \{2\}).$$

Der Transport der Strukturen ist für endliche k-Mengen U und V und eine bijektive Multifunktion $\sigma : U \to V$ als

$$(\mathcal{F} + \mathcal{G})[\sigma]\big((s, i)\big) = \begin{cases} (\mathcal{F}[\sigma], 1) & \text{für } i = 1, \\ (\mathcal{G}[\sigma], 2) & \text{für } i = 2, \end{cases}$$

definiert. Der Begriff der *summierbaren* Familie von k-sortigen Spezies und die entsprechende Summenoperation sind ebenfalls analog zum einsortigen Fall definiert.

Proposition 2.5.14. *Seien $\mathcal{F}$ und $\mathcal{G}$ zwei k-sortige Spezies, dann gilt für die erzeugenden Funktionen*

$$\begin{aligned} F + G(z_1, \ldots, z_k) &= F(z_1, \ldots, z_k) + G(z_1, \ldots, z_k) \\ \widetilde{F + G}(z_1, \ldots, z_k) &= \widetilde{F}(z_1, \ldots, z_k) + \widetilde{G}(z_1, \ldots, z_k) \\ Z_{F+G}(z_{1,1}, z_{1,2}, \ldots; \ldots; z_{k,1}, z_{k,2}, \ldots)) &= Z_F(z_{1,1}, z_{1,2}, \ldots; \ldots; z_{k,1}, z_{k,2}, \ldots) \; + \\ &\quad Z_G(z_{1,1}, z_{1,2}, \ldots; \ldots; z_{k,1}, z_{k,2}, \ldots) \end{aligned}$$

Definition 2.5.15. Seien $\mathcal{F}$ und $\mathcal{G}$ zwei k-sortige Spezies. Das *Produkt von $\mathcal{F}$ und $\mathcal{G}$*, in Zeichen $\mathcal{F} \cdot \mathcal{G}$, bildet eine endliche k-Menge U nach der Vorschrift

$$(\mathcal{F} \cdot \mathcal{G})[U] = \sum_{(U_a, U_b) \in \Delta(U)} \mathcal{F}[U_a] \times \mathcal{G}[U_b]$$

in die Menge der Strukturen ab. Der Transport der Strukturen ist für endliche k-Mengen U, V und eine bijektive Multifunktion $\sigma : U \to V$ für alle $(s, t, U_a, U_b) \in (\mathcal{F} \cdot \mathcal{G})[U]$ definiert als

$$(\mathcal{F} \cdot \mathcal{G})[\sigma]\big((s, t, U_a, U_b)\big) := \Big(\mathcal{F}\big[\sigma\big|_{U_a}\big](s), \mathcal{G}\big[\sigma\big|_{U_b}\big](t), \sigma(U_a), \sigma(U_b)\Big).$$

Proposition 2.5.16. *Seien $\mathcal{F}$ und $\mathcal{G}$ zwei k-sortige Spezies, dann gelten für die erzeugenden Funktionen der Produktspezies die Gleichungen*

$$\begin{aligned} F \cdot G(z_1, \ldots, z_k) &= F(z_1, \ldots, z_k) \cdot G(z_1, \ldots, z_k) \\ \widetilde{F \cdot G}(z_1, \ldots, z_k) &= \widetilde{F}(z_1, \ldots, z_k) \cdot \widetilde{G}(z_1, \ldots, z_k) \\ Z_{F \cdot G}(z_{1,1}, z_{1,2}, \ldots; \ldots; z_{k,1}, z_{k,2}, \ldots) &= Z_F(z_{1,1}, z_{1,2} \ldots; \ldots; z_{k,1}, z_{k,2}, \ldots) \; \cdot \\ &\quad Z_G(z_{1,1}, z_{1,2} \ldots; \ldots; z_{k,1}, z_{k,2}, \ldots). \end{aligned}$$

Definition 2.5.17. Sei $\mathcal{F}$ eine m-sortige Spezies und $(\mathcal{G}_i)_{i\in\{1,\dots,m\}}$ eine Familie von k-sortigen Spezies, sodass für alle $i \in \{1,\dots,m\}$ gilt, dass $\mathcal{G}_i[\emptyset] = \emptyset$. Die *partitionelle Komposition* $\mathcal{F}(\mathcal{G}_1,\dots,\mathcal{G}_m)$, auch als *Substitution der $\mathcal{G}_i$ in $\mathcal{F}$* bezeichnet, ist eine k-sortige Spezies, die auf jeder endlichen k-Menge $U = (U_1,\dots,U_k)$ nach der Vorschrift

$$\mathcal{F}(\mathcal{G}_1,\dots,\mathcal{G}_m)[U] = \sum_{\substack{\pi\in\mathrm{PAR}[U]\\ \chi:\pi\to[m]}} \mathcal{F}\big[\chi^{-1}\big] \times \prod_{\substack{i\in[m]\\ C\in\chi^{-1}(i)}} \mathcal{G}_i[C]$$

Strukturen erzeugt. Die m-Menge $\big(\chi^{-1}(1),\dots,\chi^{-1}(m)\big)$ wird dabei kurz als χ^{-1} bezeichnet.

Eine $\mathcal{F}(\mathcal{G}_1,\dots,\mathcal{G}_m)$-Struktur entsteht also dadurch, dass zuerst die k-sortige Menge, aus der die Struktur erzeugt wird, partitioniert wird. Die Elemente der Partition werden auf die m Sorten der Spezies $\mathcal{F}$ aufgeteilt, welche dann zur Konstruktion einer $\mathcal{F}$-Struktur verwendet werden. Sei M_i eine Element der Partition, das der i-ten Sorte zugeordnet wurde, so wird auf dieser Menge eine $\mathcal{G}_i$-Struktur erzeugt, durch die man sich die aus der Menge M_i erzeugten $\mathcal{F}$-Struktur-Komponenten ersetzt denkt.

Bemerkung. Da eine m-sortige Spezies $\mathcal{F}$ isomorph zur Spezies $\mathcal{F}(\mathcal{Z}_1,\dots,\mathcal{Z}_m)$ ist, wird, um die Anzahl der Sorten in der Notation anzugeben, für die Spezies $\mathcal{F}$ gerne der Ausdruck $\mathcal{F}(\mathcal{Z}_1,\dots,\mathcal{Z}_m)$ verwendet. Außerdem ist es üblich, analog zur Analysis die Singletons als $\mathcal{X}$, $\mathcal{Y}$, bzw. $\mathcal{Z}$ zu bezeichnen, wenn die Anzahl der Sorten klein ist.

Proposition 2.5.18. *Sei $\mathcal{F}$ eine m-sortige Spezies und $(\mathcal{G}_i)_{i\in\{1,\dots,m\}}$ eine Familie von k-sortigen Spezies, sodass für alle $i \in \{1,\dots,m\}$ gilt, dass $\mathcal{G}_i[\emptyset] = \emptyset$. Für die den Spezies zugeordneten erzeugenden Funktionen gilt dann*

$$\begin{aligned}
F(G_1,\dots,G_m)(z_1,\dots,z_k) &= F(G_1(z_1,\dots,z_k),\dots,G_m(z_1,\dots,z_k)),\\
F(\widetilde{G_1,\dots},G_m)(z_1,\dots,z_k) &= Z_F(\widetilde{G_1}(z_1,\dots,z_k),\widetilde{G_1}(z_1^2,\dots,z_k^2),\widetilde{G_1}(z_1^3,\dots,z_k^3),\dots;\\
&\qquad \widetilde{G_2}(z_1,\dots,z_k),\widetilde{G_2}(z_1^2,\dots,z_k^2),\widetilde{G_2}(z_1^3,\dots,z_k^3),\dots;\\
&\qquad \vdots\\
&\qquad \widetilde{G_m}(z_1,\dots,z_k),\widetilde{G_m}(z_1^2,\dots,z_k^2),\widetilde{G_m}(z_1^3,\dots,z_k^3),\dots),\\
Z_{F(G_1,\dots,G_m)}(z_{1,1},z_{1,2},\dots;\dots;z_{k,1},z_{k,2},\dots) &= Z_F((Z_{G_1})_1(z_{1,1},z_{1,2},\dots;\dots;z_{k,1},z_{k,2},\dots),\\
&\qquad (Z_{G_1})_2(z_{1,1},z_{1,2},\dots;\dots;z_{k,1},z_{k,2},\dots),\\
&\qquad (Z_{G_1})_3(z_{1,1},z_{1,2},\dots;\dots;z_{k,1},z_{k,2},\dots),\dots;\\
&\qquad (Z_{G_2})_1(z_{1,1},z_{1,2},\dots;\dots;z_{k,1},z_{k,2},\dots),\\
&\qquad (Z_{G_2})_2(z_{1,1},z_{1,2},\dots;\dots;z_{k,1},z_{k,2},\dots),\\
&\qquad (Z_{G_2})_3(z_{1,1},z_{1,2},\dots;\dots;z_{k,1},z_{k,2},\dots),\dots;\\
&\qquad \vdots\\
&\qquad (Z_{G_m})_1(z_{1,1},z_{1,2},\dots;\dots;z_{k,1},z_{k,2},\dots),\\
&\qquad (Z_{G_m})_2(z_{1,1},z_{1,2},\dots;\dots;z_{k,1},z_{k,2},\dots),\\
&\qquad (Z_{G_m})_3(z_{1,1},z_{1,2},\dots;\dots;z_{k,1},z_{k,2},\dots),\dots),
\end{aligned}$$

wobei $(Z_G)_l$ für $l \geq 1$ die plethystische Substitution *bezeichnet, die im mehrsortigen Fall als*

$$(Z_G)_l(z_{1,1},z_{1,2},z_{1,3}\dots;\dots;z_{k,1},z_{k,2},z_{k,3}\dots) := Z_G(z_{1,l},z_{1,2l},z_{1,3l},\dots;\dots;z_{k,l},z_{k,2l},z_{k,3l},\dots)$$

definiert ist.

Analog zum einsortigen Fall können auf mehrsortigen Spezies auch Ableitungen definiert werden. Hierbei handelt es sich allerdings um *partielle Ableitungen*, von denen es zu jeder Sorte eine gibt.

Definition 2.5.19. Sei $\mathcal{F}$ eine k-sortige Spezies, so ist für $1 \leq i \leq k$ die *i-te partielle Ableitung*, auch als $\frac{\partial}{\partial \mathcal{Z}_i}\mathcal{F}$ bezeichnet, eine k-sortige Spezies, die eine beliebige endliche k-sortige Menge $U := (U_1, \ldots, U_k)$ auf

$$\frac{\partial}{\partial \mathcal{Z}_i}\mathcal{F}[U] := \mathcal{F}\Big[\big(U_1, \ldots, U_{i-1}, U_i \cup \{*_i\}, U_{i+1}, \ldots, U_k\big)\Big]$$

abbildet, wobei $*_i$ ein Element, das nicht in der Menge U_i enthalten ist, bezeichnet, und wie im einsortigen Fall gleich U_i gesetzt werden kann.

Schließlich erweitern wir noch den für den einsortigen Fall bereits definierten Kontakt von Spezies und den Konvergenzbegriff auf den mehrsortigen Fall.

Definition 2.5.20. Seien $\mathcal{F}$ und $\mathcal{G}$ zwei k-sortige kombinatorische Spezies und $n \in \mathbb{N}_0$, dann haben $\mathcal{F}$ und $\mathcal{G}$ *Kontakt der Ordnung n*, in Zeichen $\mathcal{F} =_n \mathcal{G}$, genau dann, wenn $\mathcal{F}_{\leq n} = \mathcal{G}_{\leq n}$ gilt. Für eine k-sortige Spezies $\mathcal{F}$ bezeichnet die Spezies $\mathcal{F}_{\leq n}$ die *Einschränkung der Spezies $\mathcal{F}$ auf Multimengen der totalen Kardinaliät n*, die auf einer endlichen Multimenge $U = (U_1, \ldots, U_k)$ die Strukturen

$$\mathcal{F}_{\leq n}[U_1, \ldots, U_k] = \begin{cases} \mathcal{F}[U_1, \ldots, U_k] & \text{für } \|U\| \leq n, \\ \emptyset & \text{sonst,} \end{cases}$$

bildet, und deren Transport der Strukturen für eine weitere endliche Multimenge $V = (V_1, \ldots, V_k)$ und eine beliebige bijektive Multifunktion $\sigma = (\sigma_1, \ldots, \sigma_k) : (U_1, \ldots, U_k) \to (V_1, \ldots, V_k)$ als

$$\mathcal{F}_{\leq n} = \begin{cases} \mathcal{F}[\sigma] & \text{für } \|U\| \leq n, \\ f_\emptyset & \text{sonst,} \end{cases}$$

definiert ist.

Definition 2.5.21. Eine Folge von k-sortigen Spezies $\big(\mathcal{F}_n\big)_{n\in\mathbb{N}_0}$ konvergiert genau dann gegen die k-sortige Spezies $\mathcal{F}$, wenn für alle $N \in \mathbb{N}$ ein $K \in \mathbb{N}$ existiert, sodass für alle $n \geq K$ gilt, dass $\mathcal{F}_n =_N \mathcal{F}$. Wir schreiben dann

$$\lim_{n\to\infty} \mathcal{F}_n = \mathcal{F}.$$

2.6 Kombinatorische Systeme

In der Praxis haben wir es meistens mit kombinatorischen Spezifikationen, also Definitionen von kombinatorischen Strukturen zu tun, die durch Anwendung der vorgestellten Operationen auf die Basisspezies entstehen. Diese Spezifikationen sind in Form eines Gleichungssystems gegeben. Wir werden nun untersuchen, wie wir diese Spezifikationen zu interpretieren haben, und unter welchen Bedingunen solche Systeme überhaupt eine kombinatorisch sinnvolle Spezifikation darstellen. Wir beginnen die Untersuchung mit ein paar Definitionen.

Definition 2.6.1. Sei $n \in \mathbb{N}$ und seien $\mathcal{F}_1, \ldots, \mathcal{F}_n$ k-sortige Spezies, dann wird die Familie $\big(\mathcal{F}_i\big)_{i\in\{1,\ldots,n\}}$ als *n-dimensionaler Vektor von k-sortigen Spezies* bezeichnet. Wir schreiben für diese Familie auch $\boldsymbol{\mathcal{F}}$ beziehungsweise $\big(\mathcal{F}_{1:n}\big)$ oder $\big(\mathcal{F}_1, \ldots, \mathcal{F}_n\big)$. Weiters heißt eine Familie $\big(\mathcal{M}_{i,j}\big)_{\substack{1\leq i\leq n\\ 1\leq j\leq m}}$ von k-sortigen Spezies eine *$n \times m$-Matrix von k-sortigen Spezies*, wobei *n Zeilenanzahl* und *m Spaltenanzahl* genannt wird. Diese Matrix wird auch mit $\boldsymbol{\mathcal{M}}$ beziehungsweise bei aus dem Kontext bekannter Zeilen- und Spaltenanzahl als $\big(\mathcal{M}_{i,j}\big)$ bezeichnet. Eine Matrix heißt *quadratisch*, wenn die Anzahl der Zeilen gleich der Anzahl der Spalten ist. Seien $\boldsymbol{\mathcal{A}} = \big(\mathcal{A}_{i,j}\big)$ und $\boldsymbol{\mathcal{B}} = \big(\mathcal{B}_{i,j}\big)$ zwei $n \times m$-Matrizen von k-sortigen Spezies, dann ist die Summe $\boldsymbol{\mathcal{C}} := \boldsymbol{\mathcal{A}} + \boldsymbol{\mathcal{B}}$ ebenfalls eine $n \times m$-Matrix, die als $\mathcal{C}_{i,j} := \mathcal{A}_{i,j} + \mathcal{B}_{i,j}$ für alle $1 \leq i \leq n$ und alle $1 \leq j \leq m$ definiert ist. Weiters definieren wir das Produkt $\boldsymbol{\mathcal{A}}\cdot\boldsymbol{\mathcal{B}}$ einer $n\times m$-Matrix $\boldsymbol{\mathcal{A}} := \big(\mathcal{A}_{i,j}\big)$ und einer $m\times l$-Matrix $\boldsymbol{\mathcal{B}} := \big(\mathcal{B}_{i,j}\big)$ k-sortiger Spezies als jene $n\times l$-Matrix $\boldsymbol{\mathcal{C}} := \big(\mathcal{C}_{i,j}\big)$, für die $\mathcal{C}_{i,j} = \sum_{k=1}^{m} \mathcal{A}_{i,k}\cdot\mathcal{B}_{k,j}$ für alle $1 \leq i \leq n$

und alle $1 \leq j \leq l$ gilt. Die *Einheitsmatrix* $\mathbf{Id}$ ist jene quadratische Matrix, deren Diagonale aus der $\mathbf{1}$-Spezies, und deren restliche Einträge aus der $\mathbf{0}$-Spezies bestehen. Die *Nullmatrix* $\mathbf{0}$ ist jene Matrix, deren Einträge alle aus der $\mathbf{0}$-Spezies bestehen. Die Potenzen einer quadratischen Matrix $\boldsymbol{\mathcal{A}}$ ist rekursiv als $\boldsymbol{\mathcal{A}}^0 := \mathbf{Id}$ und $\boldsymbol{\mathcal{A}}^{n+1} := \boldsymbol{\mathcal{A}}^n \cdot \boldsymbol{\mathcal{A}}$ für alle $n \in \mathbb{N}_0$ definiert. Schließlich heißt eine quadratische Matrix $\boldsymbol{\mathcal{A}}$ genau dann *nilpotent*, wenn ein $n \in \mathbb{N}$ existiert, sodass $\boldsymbol{\mathcal{A}}^n = \mathbf{0}$. Falls die Matrix $\boldsymbol{\mathcal{A}}$ nilpotent ist, wird das kleinste $n \in \mathbb{N}$, sodass $\boldsymbol{\mathcal{A}}^n = \mathbf{0}$ ist, als *Index der Nilpotenz* bezeichnet.

Bemerkung. Viele der aus der linearen Algebra bekannten Eigenschaften der Matrizen-Operationen übertragen sich, wie wir leicht nachrechnen können, auf die Operationen auf Matrizen von Spezies. Dazu zählen unter anderem das Assoziativitätsgesetz und die Kommutativität der Addition. Die Nullmatrix ist ein neutrales Element bezüglich der Addition, die Einheitsmatrix ein neutrales Element bezüglich der Multiplikation.

Da wir in Zukunft auch Folgen von Vektoren von Spezies betrachten werden, übertragen wir dafür den Konvergenzbegriff von Spezies.

Definition 2.6.2. Sei $\left(\boldsymbol{\mathcal{F}}^{[n]}\right)_{n\in\mathbb{N}}$ eine Folge von m-dimensionalen Vektoren von k-sortigen Spezies und $\boldsymbol{\mathcal{F}} = \left(\mathcal{F}_{1:m}\right)$ ein m-dimensionaler Vektor von k-sortigen Spezies, dann konvergiert die Folge $\left(\boldsymbol{\mathcal{F}}^{[n]}\right)_{n\in\mathbb{N}}$ genau dann gegen den Vektor $\boldsymbol{\mathcal{F}}$, in Zeichen $\lim_{n\to\infty} \boldsymbol{\mathcal{F}}^{[n]} = \boldsymbol{\mathcal{F}}$, wenn alle Komponenten konvergieren, wenn also für alle $i \in \{1, \ldots, m\}$ gilt, dass $\lim_{n\to\infty} \mathcal{F}_i^{[n]} = \mathcal{F}_i$. Seien weiters $\boldsymbol{\mathcal{F}} = \left(\mathcal{F}_{1:m}\right)$ und $\boldsymbol{\mathcal{G}} = \left(\mathcal{G}_{1:m}\right)$ zwei m-dimensionale Vektoren von k-sortigen Spezies, dann haben sie *Kontakt der Ordnung* n, in Zeichen $\boldsymbol{\mathcal{F}} =_n \boldsymbol{\mathcal{G}}$ genau dann, wenn alle Komponenten Kontakt der Ordnung n haben, wenn also für alle $i \in \{1, \ldots, m\}$ gilt, dass $\mathcal{F}_i =_n \mathcal{G}_i$.

Definition 2.6.3. Sei $\boldsymbol{\mathcal{H}} := \left(\mathcal{H}_{1:m}\right)$ ein Vektor von $m+k$ sortigen Spezies. Ein *kombinatorisches System* ist ein Gleichungssystem der Form $\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{H}}\left(\boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{Y}}\right) = \boldsymbol{\mathcal{H}}\left(\mathcal{Z}_1, \ldots, \mathcal{Z}_k, \mathcal{Y}_1, \ldots, \mathcal{Y}_m\right)$. Die Lösungen $\boldsymbol{\mathcal{Y}} = \left(\mathcal{Y}_{1:m}\right) = \left(\mathcal{Y}_1(\mathcal{Z}_1, \ldots, \mathcal{Z}_k), \ldots, \mathcal{Y}_m(\mathcal{Z}_1, \ldots, \mathcal{Z}_k)\right)$ dieses Systems sind jene k-sortigen Spezies-Vektoren, die durch das System spezifiziert werden.

Nun können wir uns die Fragen stellen, unter welchen Bedingungen Lösungen existieren, wann diese Lösungen eindeutig sind, und wie sie, falls sie existieren, konstruiert werden können. Diese Fragen werden im Artikel [14], der auf [9] und [2] aufbaut, zufriedenstellend beantwortet. Der Rest dieses Abschnitts ist eine Zusammenfassung der wichtigsten Ergebnisse.

Die Untersuchungen betrachten zunächst Systeme, für die $\boldsymbol{\mathcal{H}}\left(\mathbf{0}, \mathbf{0}\right) = \mathbf{0}$ gilt. Der definierende Spezies-Vektor $\boldsymbol{\mathcal{H}}$ erzeugt also auf der leeren Spezies keine Strukturen. In diesem Fall gilt der in [9] zuerst bewiesene *Satz der impliziten Spezies*. Dieser setzt noch voraus, dass die sogenannte *Jacobi-Matrix* von $\boldsymbol{\mathcal{H}}$ nilpotent ist. Diese ist wie folgt definiert.

Definition 2.6.4. Sei $\boldsymbol{\mathcal{H}} = \left(\mathcal{H}_{1:m}\right)$ ein Vektor von $m+k$-sortigen Spezies und $\boldsymbol{\mathcal{Y}} = \left(\mathcal{Y}_{1:m}\right)$ ein Vektor von k-sortigen Spezies. Die *Jacobi-Matrix* des Vektors $\boldsymbol{\mathcal{H}}(\boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{Y}}) = \boldsymbol{\mathcal{H}}(\mathcal{Z}_1, \ldots, \mathcal{Z}_k, \mathcal{Y}_1, \ldots, \mathcal{Y}_m)$ bezüglich $\boldsymbol{\mathcal{Y}}$, auch als $\boldsymbol{\partial\mathcal{H}/\partial\mathcal{Y}}$ bezeichnet, ist jene $m \times m$-Matrix, deren Eintrag (i, j) für alle $1 \leq i, j \leq m$ gleich $\partial\mathcal{H}_i(\boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{Y}})/\partial\mathcal{Y}_j$ ist.

Satz 2.6.5 (Satz der impliziten Spezies)**.** *Sei* $\boldsymbol{\mathcal{H}} = \boldsymbol{\mathcal{H}}\left(\boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{Y}}\right) = \boldsymbol{\mathcal{H}}\left(\mathcal{Z}_1, \ldots, \mathcal{Z}_k, \mathcal{Y}_1, \ldots, \mathcal{Y}_m\right)$ *ein* m*-dimensionaler Vektor von* $m+k$ *sortigen Spezies mit* $\boldsymbol{\mathcal{H}}\left(\mathbf{0}, \mathbf{0}\right) = \mathbf{0}$ *und einer nilpotenten Jacobi-Matrix* $\boldsymbol{\partial\mathcal{H}/\partial\mathcal{Y}}$*, dann existiert eine bis auf Isomorphie eindeutige Lösung* $\boldsymbol{\mathcal{S}}$ *mit* $\boldsymbol{\mathcal{S}}(\mathbf{0}) = \mathbf{0}$ *des Systems*

$$\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{H}}\left(\boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{Y}}\right) \text{ mit } \boldsymbol{\mathcal{Y}} = \left(\mathcal{Y}_{1:m}\right).$$

Dieser Satz wird in [14, S. 12-13] bewiesen, indem ausgehend von der leeren Spezies eine Iterationsfolge definiert wird und deren Konvergenz gezeigt wird.

Satz 2.6.6. *Sei* $\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{H}}\left(\boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{Y}}\right)$ *ein kombinatorisches System, sodass* $\boldsymbol{\mathcal{H}}\left(\mathbf{0}, \mathbf{0}\right) = \mathbf{0}$ *und die Matrix* $\boldsymbol{\partial\mathcal{H}/\partial\mathcal{Y}}\left(\mathbf{0}, \mathbf{0}\right)$ *nilpotent ist. Die Folge* $\left(\boldsymbol{\mathcal{Y}}^{[n]}\right)_{n\in\mathbb{N}}$ *die durch*

$$\boldsymbol{\mathcal{Y}}^{[0]} := \mathbf{0} \text{ und } \boldsymbol{\mathcal{Y}}^{[n+1]} := \boldsymbol{\mathcal{H}}\left(\boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{Y}}^{[n]}\right) \text{ für alle } n \geq 0$$

definiert ist, ist konvergent.

Es existiert also eine Spezies-Vektor $\boldsymbol{\mathcal{Y}}$, sodass einerseits $\lim_{n\to\infty} \boldsymbol{\mathcal{Y}}^{[n]} = \boldsymbol{\mathcal{Y}}$, andererseits aber auch $\lim_{n\to\infty} \boldsymbol{\mathcal{H}}\big(\boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{Y}}^{[n]}\big) = \lim_{n\to\infty} \boldsymbol{\mathcal{Y}}^{[n+1]} = \boldsymbol{\mathcal{Y}}$ gilt. Aus dem folgenden Lemma können wir also auf die Existenz einer Lösung des kombinatorischen Systems $\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{H}}\big(\boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{Y}}\big)$ schließen.

Lemma 2.6.7. *Sei $\boldsymbol{\mathcal{H}}\big(\boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{Y}}\big) = \boldsymbol{\mathcal{H}}\big(\mathcal{Z}_1, \ldots, \mathcal{Z}_k, \mathcal{Y}_1, \ldots, \mathcal{Y}_m\big)$ ein m-dimensionaler Vektor von $m+k$ sortigen Spezies und $\big(\boldsymbol{\mathcal{S}}^{[n]}\big)_{n\in\mathbb{N}}$ eine Folge von m-dimensionalen Vektoren k-sortiger Spezies, die gegen $\boldsymbol{\mathcal{S}}$ konvergiert. Wenn die Folge $\boldsymbol{\mathcal{H}}\big(\boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{S}}^{[n]}\big)$ ebenfalls gegeben $\boldsymbol{\mathcal{S}}$ konvergiert, dann ist $\boldsymbol{\mathcal{S}}$ eine Lösung des Systems $\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{H}}\big(\boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{Y}}\big)$.*

Mit dem folgenden Lemma wird einerseits die Konvergenz der Iterationsfolge, andererseits durch Anwendung von Lemma 2.6.7 und vollständige Induktion die Eindeutigkeit dieser Lösung bis auf Isomorphie gezeigt. Im Folgenden bezeichnen wir die i-te Iteration von $\boldsymbol{\mathcal{Y}} \mapsto \boldsymbol{\mathcal{H}}\big(\boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{Y}}\big)$ mit $\boldsymbol{\mathcal{H}}^i$.

Lemma 2.6.8. *Sei $\boldsymbol{\mathcal{H}}\big(\boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{Y}}\big) = \boldsymbol{\mathcal{H}}\big(\mathcal{Z}_1, \ldots, \mathcal{Z}_k, \mathcal{Y}_1, \ldots \mathcal{Y}_m\big)$ ein m-dimensionaler Vektor von $m+k$-sortigen Spezies, sodass $\boldsymbol{\mathcal{H}}\big(\mathbf{0}, \mathbf{0}\big) = \mathbf{0}$ und die Jacobi-Matrix $\boldsymbol{\partial\mathcal{H}}/\boldsymbol{\partial\mathcal{Y}}\big(\mathbf{0}, \mathbf{0}\big)$ nilpotent ist. Seien weiters $\boldsymbol{\mathcal{A}}$ und $\boldsymbol{\mathcal{B}}$ zwei m-dimensionale Vektoren k-sortiger Spezies mit $\boldsymbol{\mathcal{B}} \subset \boldsymbol{\mathcal{A}}$, dann folgt aus $\boldsymbol{\mathcal{B}} =_n \boldsymbol{\mathcal{A}}$, dass $\boldsymbol{\mathcal{H}}^p\big(\boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{B}}\big) =_{n+1} \boldsymbol{\mathcal{H}}^p\big(\boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{A}}\big)$, wobei $1 \leq p \leq m$ den Index der Nilpotenz der Jacobi-Matrix bezeichnet.*

Zur Konstruktion von Lösungen eines kombinatorischen Systems werden wir in Folge immer Iterationsfolgen betrachten. Dies führt uns zur Definition der sogenannten *wohlfundierten* Systeme. Es ist sinnvoll, wenn wir nur Systeme betrachten, deren Lösung keine Koordinaten besitzen, die keine Strukturen erzeugen.

Definition 2.6.9. Sei $\boldsymbol{\mathcal{S}} = \big(\mathcal{S}_{1:n}\big)$ ein n-dimensionaler Vektor von k-sortigen Spezies. Ein Index i wird genau dann *Nullkoordinate* des Vektors $\boldsymbol{\mathcal{S}}$ genannt, wenn $\mathcal{S}_i = \mathbf{0}$.

Definition 2.6.10. Sei $\boldsymbol{\mathcal{H}}\big(\boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{Y}}\big) = \boldsymbol{\mathcal{H}}\big(\mathcal{Z}_1, \ldots, \mathcal{Z}_k, \mathcal{Y}_1, \ldots, \mathcal{Y}_m\big)$ ein m-dimensionaler Vektor von $m + k$-sortigen Spezies, sodass $\boldsymbol{\mathcal{H}}\big(\mathbf{0}, \mathbf{0}\big) = \mathbf{0}$. Das kombinatorische System $\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{H}}\big(\boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{Y}}\big)$ heißt genau dann *wohlfundiert bei* $\mathbf{0}$, wenn die Folge $\big(\boldsymbol{\mathcal{Y}}^{[n]}\big)_{n\in\mathbb{N}_0}$, die durch

$$\boldsymbol{\mathcal{Y}}^{[0]} := \mathbf{0} \text{ und } \boldsymbol{\mathcal{Y}}^{[n+1]} := \boldsymbol{\mathcal{H}}\big(\boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{Y}}^{[n]}\big) \text{ für } n \geq 0$$

definiert ist, konvergiert und der Grenzwert $\boldsymbol{\mathcal{S}}$ dieser Folge keine Nullkoordinate besitzt.

Tatsächlich gibt es ein Kriterium für die Wohlfundiertheit eines Systems bei $\mathbf{0}$, dessen Beweis in [14, S. 15] zu finden ist.

Satz 2.6.11. *Sei $\boldsymbol{\mathcal{H}}\big(\boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{Y}}\big) = \boldsymbol{\mathcal{H}}\big(\mathcal{Z}_1, \ldots, \mathcal{Z}_k, \mathcal{Y}_1, \ldots, \mathcal{Y}_m\big)$ ein m-dimensionaler Vektor von $m+k$-sortigen Spezies, sodass $\boldsymbol{\mathcal{H}}\big(\mathbf{0}, \mathbf{0}\big) = \mathbf{0}$. Das kombinatorische System $\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{H}}\big(\boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{Y}}\big)$ ist genau dann wohlfundiert bei $\mathbf{0}$, wenn die Jacobi-Matrix $\boldsymbol{\partial\mathcal{H}}/\boldsymbol{\partial\mathcal{Y}}\big(\mathbf{0}, \mathbf{0}\big)$ nilpotent ist und der durch die in Definition 2.6.10 angegebene Iterationsfolge definierte Spezies-Vektor $\boldsymbol{\mathcal{Y}}^{[m]}$ keine Nullkoordinate besitzt.*

Beispiel 2.6.12. Das durch den Spezies-Vektor

$$\begin{aligned}\mathcal{H}_1\big(\mathcal{Z}, \mathcal{B}, \mathcal{C}\big) &= \mathcal{Z} + \mathcal{C}\\ \mathcal{H}_2\big(\mathcal{Z}, \mathcal{B}, \mathcal{C}\big) &= \mathcal{B} \cdot \mathcal{Z} \cdot \mathcal{B}\end{aligned}$$

gegebene System $\boldsymbol{\mathcal{H}}\big(\mathcal{Z}, \boldsymbol{\mathcal{Y}}\big) = \boldsymbol{\mathcal{Y}}$ spezifiziert Binärbäume ohne Unterscheidung zwischen internen und externe Knoten. Dieses System ist wohlfundiert bei $\mathbf{0}$, weil

$$\begin{aligned}\mathcal{H}_1\big(\mathbf{0}, \mathbf{0}, \mathbf{0}\big) &= \mathbf{0} + \mathbf{0} = \mathbf{0} \text{ und}\\ \mathcal{H}_2\big(\mathbf{0}, \mathbf{0}, \mathbf{0}\big) &= \mathbf{0} \cdot \mathbf{0} \cdot \mathbf{0} = \mathbf{0}\end{aligned}$$

gilt, und die für die Jacobi-Matrix $\boldsymbol{\partial\mathcal{H}/\partial\mathcal{Y}}$, die aus den partiellen Ableitungen

$$\begin{aligned}
\partial\mathcal{H}_1/\partial\mathcal{B} &= \frac{\partial\mathcal{Z}}{\partial\mathcal{B}} + \frac{\partial\mathcal{C}}{\partial\mathcal{B}} = \mathbf{0} + \mathbf{0} = \mathbf{0},\\
\partial\mathcal{H}_1/\partial\mathcal{C} &= \frac{\partial\mathcal{Z}}{\partial\mathcal{C}} + \frac{\partial\mathcal{Z}}{\partial\mathcal{Z}} = \mathbf{0} + \mathbf{1} = \mathbf{1},\\
\partial\mathcal{H}_2/\partial\mathcal{B} &= \underbrace{\frac{\partial\mathcal{B}}{\partial\mathcal{B}}}_{\mathbf{1}}\cdot\mathcal{Z}\cdot\mathcal{B} + \mathcal{B}\cdot\underbrace{\frac{\partial\mathcal{Z}}{\partial\mathcal{B}}}_{\mathbf{0}} + \mathcal{B}\cdot\mathcal{Z}\cdot\underbrace{\frac{\partial\mathcal{B}}{\partial\mathcal{B}}}_{\mathbf{1}} = \mathcal{Z}\cdot\mathcal{B} + \mathcal{B}\cdot\mathcal{Z} \text{ und}\\
\partial\mathcal{H}_2/\partial\mathcal{C} &= \underbrace{\frac{\partial\mathcal{B}}{\partial\mathcal{C}}}_{\mathbf{0}}\cdot\mathcal{Z}\cdot\mathcal{B} + \mathcal{B}\cdot\underbrace{\frac{\partial\mathcal{Z}}{\partial\mathcal{C}}}_{\mathbf{0}} + \mathcal{B}\cdot\mathcal{Z}\cdot\underbrace{\frac{\partial\mathcal{B}}{\partial\mathcal{C}}}_{\mathbf{0}} = \mathbf{0}
\end{aligned}$$

besteht, gilt, dass

$$\frac{\boldsymbol{\partial\mathcal{H}}}{\boldsymbol{\partial\mathcal{Y}}}(\mathbf{0},\mathbf{0}) = \begin{pmatrix}\mathbf{0} & \mathbf{1}\\ \mathbf{0} & \mathbf{0}\end{pmatrix}.$$

Die Jacobi-Matrix ist an der Stelle $(\mathbf{0},\mathbf{0})$ also nilpotent. Schließlich erkennen wir noch, dass

$$\begin{aligned}
\boldsymbol{\mathcal{Y}}^{[1]} &= \boldsymbol{\mathcal{H}}(\mathcal{Z},\mathbf{0},\mathbf{0}) = \begin{pmatrix}\mathcal{Z}\\ \mathbf{0}\end{pmatrix} \text{ und}\\
\boldsymbol{\mathcal{Y}}^{[2]} &= \boldsymbol{\mathcal{H}}(\mathcal{Z},\mathcal{Z},\mathbf{0}) = \begin{pmatrix}\mathcal{Z}\\ \mathcal{Z}\cdot\mathcal{Z}\cdot\mathcal{Z}\end{pmatrix},
\end{aligned}$$

und daher keine Nullkoordinaten in der Lösung existieren.

Leider ist die Wohlfundiertheit bei $\mathbf{0}$ nicht allgemein genug, um alle in der Praxis vorkommende kombinatorische Systeme behandeln zu können, wie folgendes Beispiel zeigt.

Beispiel 2.6.13. Das System, das Binärbäume spezifiziert, die aus internen und externen Knoten bestehen, sodass die internen Knoten eine Marke tragen, ist durch den Spezies-Vektor $\boldsymbol{\mathcal{H}}(\mathcal{Z},\mathcal{B},\mathcal{C})$

$$\begin{aligned}
\mathcal{H}_1(\mathcal{Z},\mathcal{B},\mathcal{C}) &= \mathbf{1} + \mathcal{C}\\
\mathcal{H}_2(\mathcal{Z},\mathcal{B},\mathcal{C}) &= \mathcal{B}\cdot\mathcal{Z}\cdot\mathcal{B}
\end{aligned}$$

definiert. Die durch das System bestimmte Iterationsfolge konvergiert, allerdings ist dieses System nicht wohlfundiert bei $\mathbf{0}$, da

$$\boldsymbol{\mathcal{H}}(\mathbf{0},\mathbf{0},\mathbf{0}) = \begin{pmatrix}\mathbf{1}\\ \mathbf{0}\end{pmatrix} \neq \begin{pmatrix}\mathbf{0}\\ \mathbf{0}\end{pmatrix}.$$

Um auch solche System behandeln können, kann eine in [14] präsentierte Verallgemeinerung des Satzes der impliziten Spezies herangezogen werden. Dieser Satz und die Definitionen beziehungsweise Resultate, die zum Beweis dieses Satzes benötigt werden, werden nun zitiert.

Definition 2.6.14. Eine (mehrsortige) Spezies von Strukturen $\mathcal{F}$ heißt genau dann *polynomiell*, wenn ein $n \geq 0$ existiert, sodass $\mathcal{F}_{\geq n} = \mathbf{0}$, wenn also nur endlich viele $\mathcal{F}$-Strukturen existieren.

Für bei $\mathbf{0}$ wohlfundierte Systeme gibt es ein Kriterium für die Polynomialität der Lösungen. Wie in [14, S. 16] bewiesen gilt der folgende Satz.

Satz 2.6.15 (Implizite Polynomielle Spezies)**.** *Sei $\boldsymbol{\mathcal{H}}(\boldsymbol{\mathcal{Z}},\boldsymbol{\mathcal{Y}}) = \boldsymbol{\mathcal{H}}(\mathcal{Z}_1,\ldots,\mathcal{Z}_k,\mathcal{Y}_1,\ldots,\mathcal{Y}_m)$ ein m-dimensionaler Vektor von $m+k$-sortigen Spezies, sodass $\boldsymbol{\mathcal{H}}(\mathbf{0},\mathbf{0}) = \mathbf{0}$ und die Jacobi-Matrix $\boldsymbol{\partial\mathcal{H}/\partial\mathcal{Y}}(\mathbf{0},\mathbf{0})$ nilpotent ist. Sei weiters $(\boldsymbol{\mathcal{Y}}^{[n]})$ jene Folge von Spezies-Vektoren, die durch*

$$\boldsymbol{\mathcal{Y}}^{[0]} = \mathbf{0} \text{ und } \boldsymbol{\mathcal{Y}}^{[n+1]} = \boldsymbol{\mathcal{H}}(\boldsymbol{\mathcal{Z}},\boldsymbol{\mathcal{Y}}^{[n]}) \text{ für } n \geq 0$$

definiert ist. Die Lösung $\boldsymbol{\mathcal{S}}$ des Systems $\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{H}}(\boldsymbol{\mathcal{Z}},\boldsymbol{\mathcal{Y}})$, sodass $\boldsymbol{\mathcal{S}}(\mathbf{0}) = \mathbf{0}$ ist genau dann polynomiell, wenn $\boldsymbol{\mathcal{Y}}^{[m]}$ polynomiell ist und $\boldsymbol{\mathcal{Y}}^{[m]} = \boldsymbol{\mathcal{Y}}^{[m+1]}$.

Sei nun $\mathcal{F}\big(\boldsymbol{\mathcal{Z}}_1, \boldsymbol{\mathcal{Z}}_2\big) = \mathcal{F}\big(\mathcal{Z}_{1,1}, \ldots, \mathcal{Z}_{1,m_1}, \mathcal{Z}_{2,1}, \ldots, \mathcal{Z}_{2,m_2}\big)$ eine $m_1 + m_2$-sortige Spezies. Für eine beliebige $\mathcal{F}$-Struktur $s \in \mathcal{F}\big[U_{1,1}, \ldots, U_{1,m_1}, U_{2,1}, \ldots, U_{2,m_2}\big]$ scheiben wir in Folge $|s|_1 := \big\|\big(U_{1,1}, \ldots, U_{1,m_1}\big)\big\|$ und $|s|_2 := \big\|\big(U_{2,1}, \ldots, U_{2,m_2}\big)\big\|$. Weiters bezeichnen wir mit $\mathcal{F}_{=(k,n)}$ jene Unterspezies von $\mathcal{F}$, sodass für alle $s \in \mathcal{F}_{=(k,n)}$ gilt, dass $|s|_1 = k$ und $|s|_2 = n$.

Definition 2.6.16. Eine m_1+m_2 sortige Spezies $\mathcal{F}\big(\boldsymbol{\mathcal{Z}}_1, \boldsymbol{\mathcal{Z}}_2\big) = \mathcal{F}\big(\mathcal{Z}_{1,1}, \ldots, \mathcal{Z}_{1,m_1}, \mathcal{Z}_{2,1}, \ldots, \mathcal{Z}_{2,m_2}\big)$ ist polynomiell in den Sorten $\boldsymbol{\mathcal{Z}}_1$ genau dann, wenn die Spezies $\mathcal{F}_{=(\cdot,n)} := \sum_{k\geq 0} \mathcal{F}_{=(k,n)}$ für alle $n \geq 0$ polynomiell ist.

Satz 2.6.17 (Implizite partiell polynomielle Spezies)**.**
Sei $\boldsymbol{\mathcal{H}}\big(\mathcal{Z}_1, \boldsymbol{\mathcal{Z}}_2, \boldsymbol{\mathcal{Y}}\big) = \boldsymbol{\mathcal{H}}\big(\mathcal{Z}_1, \mathcal{Z}_{2,1}, \ldots, \mathcal{Z}_{2,k}, \mathcal{Y}_1, \ldots, \mathcal{Y}_m\big)$ ein m-dimensionaler Vektor von $m+k+1$-sortigen Spezies, sodass das System $\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{H}}\big(\mathcal{Z}_1, \boldsymbol{\mathcal{Z}}_2, \boldsymbol{\mathcal{Y}}\big)$ wohlfundiert bei $\mathbf{0}$ ist, und sei $\boldsymbol{\mathcal{S}}\big(\mathcal{Z}_1, \boldsymbol{\mathcal{Z}}_2\big)$ dessen Lösung, sodass $\boldsymbol{\mathcal{S}}\big(\mathbf{0}\big) = \mathbf{0}$. Die Spezies $\boldsymbol{\mathcal{S}}\big(\mathcal{Z}_1, \boldsymbol{\mathcal{Z}}_2\big)$ ist polynomiell in $\mathcal{Z}_1$ genau dann, wenn

1. *die Spezies $\boldsymbol{\mathcal{S}}_0\big(\mathcal{Z}_1\big) := \boldsymbol{\mathcal{S}}\big(\mathcal{Z}_1, \mathbf{0}\big)$ polynomiell ist,*
2. *die Jacobi-Matrix $\boldsymbol{\partial\mathcal{H}}/\boldsymbol{\partial\mathcal{Y}}\big(\mathcal{Z}_1, \mathbf{0}, \boldsymbol{\mathcal{S}}_0\big(\mathcal{Z}_1\big)\big)$ nilpotent ist,*
3. *$\boldsymbol{\mathcal{H}}$ polynomiell in $\mathcal{Z}_1$ ist, und für alle $i \in \{1, \ldots, m\}$ gilt, dass entweder die i-te Koordinate von $\boldsymbol{\mathcal{S}}_0$ gleich $\mathbf{0}$ ist oder $\boldsymbol{\mathcal{H}}\big(\mathcal{Z}_1, \boldsymbol{\mathcal{Z}}_2, \boldsymbol{\mathcal{Y}}\big)$ polynomiell in $\mathcal{Y}_i$ ist.*

Nun können wir den Begriff der Wohlfundiertheit auf eine allgemeinere Klasse von Systemen erweitern.

Definition 2.6.18. Sei $\boldsymbol{\mathcal{H}}\big(\boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{Y}}\big) = \boldsymbol{\mathcal{H}}\big(\mathcal{Z}_1, \ldots, \mathcal{Z}_k, \mathcal{Y}_1, \ldots, \mathcal{Y}_m\big)$ ein m-dimensionaler Vektor von $m+k$-sortigen Spezies. Das kombinatorische System $\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{H}}\big(\boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{Y}}\big)$ heißt genau dann *wohlfundiert*, wenn die Iteration

$$\boldsymbol{\mathcal{Y}}^{[0]} = \mathbf{0} \text{ und } \boldsymbol{\mathcal{Y}}^{[n+1]} = \boldsymbol{\mathcal{H}}\big(\boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{Y}}^{[n]}\big) \text{ für alle } n \geq 0,$$

wohldefiniert ist, eine konvergente Folge definiert und der Grenzwert $\boldsymbol{\mathcal{S}}$ dieser Folge keine Nullkoordinaten besitzt. Wohldefiniert bedeutet in diesem Kontext, dass die Komposition der Spezies tatsächlich definiert ist, dass also für jede Sorte $\mathcal{Y}_i$ gilt, dass entweder $\boldsymbol{\mathcal{H}}$ polynomiell in $\mathcal{Y}_i$ ist, oder $\mathcal{Y}_i^{[n]}\big(\mathbf{0}\big) = \mathbf{0}$ für alle $n \geq 0$.

Wie für die Wohlfundiertheit bei $\mathbf{0}$ gibt es auch für die Wohlfundiertheit eines Systems ein Kriterium (siehe [14, S. 21]).

Definition 2.6.19. Sei $\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{H}}\big(\boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{Y}}\big)$ ein System, sodass $\boldsymbol{\mathcal{H}}\big(\mathbf{0}, \mathbf{0}\big) \neq \mathbf{0}$. Das *genullte Erweiterungssystem* ist als

$$\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{K}}\big(\mathcal{Z}_1, \boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{Y}}\big) \text{ definiert, wobei } \boldsymbol{\mathcal{K}} = \boldsymbol{\mathcal{H}}\big(\boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{Y}}\big) - \boldsymbol{\mathcal{H}}\big(\mathbf{0}, \mathbf{0}\big) + \mathcal{Z}_1\boldsymbol{\mathcal{H}}\big(\mathbf{0}, \mathbf{0}\big) \text{ gilt.}$$

Satz 2.6.20 (Charakterisierung wohlfundierter Systeme)**.** *Sei $\boldsymbol{\mathcal{H}}\big(\boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{Y}}\big)$ ein m-dimensionaler Vektor von $m+k$-sortigen Spezies. Das kombinatorische System $\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{H}}\big(\boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{Y}}\big)$ ist genau dann wohlfundiert, wenn gilt, dass*

1. *das genullte Erweiterungssystem $\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{K}}\big(\mathcal{Z}_1, \boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{Y}}\big)$ bei $\mathbf{0}$ wohlfundiert ist, und falls außerdem gilt, dass*
2. *$\boldsymbol{\mathcal{S}}_1\big(\mathcal{Z}_1, \boldsymbol{\mathcal{Z}}\big)$ polynomiell in $\mathcal{Z}_1$ ist, falls die Spezies $\boldsymbol{\mathcal{S}}_1\big(\mathcal{Z}_1, \boldsymbol{\mathcal{Z}}\big)$ eine Lösung des erweiterten Systems $\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{K}}\big(\mathcal{Z}_1, \boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{Y}}\big)$ mit $\boldsymbol{\mathcal{S}}_1\big(0, \mathbf{0}\big) = \mathbf{0}$ ist.*

In diesem Fall ist $\boldsymbol{\mathcal{S}}_1\big(1, \boldsymbol{\mathcal{Z}}\big)$ der Grenzwert der in Definition 2.6.18 angegebenen Iterationsfolge.

Nun können wir mit dem verallgemeinerten Satz der impliziten Spezies das Hauptergebnis dieses Abschnitts angeben. Mit den bereits vorgestellten Sätzen stehen uns Mittel zur Verfügung, um dessen Voraussetzungen für gegebene kombinatorische Systeme algorithmisch zu überprüfen.

Satz 2.6.21 (Verallgemeinerter Satz der impliziten Spezies)**.**
Sei $\boldsymbol{\mathcal{H}}\big(\boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{Y}}\big) = \boldsymbol{\mathcal{H}}\big(\mathcal{Z}_1, \ldots, \mathcal{Z}_k, \mathcal{Y}_1, \ldots, \mathcal{Y}_m\big)$ ein m-dimensionaler Vektor von $m+k$-sortigen Spezies. Wenn das System $\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{H}}\big(\boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{Y}}\big)$ wohlfundiert ist, dann existiert eine bis auf Isomorphie eindeutige Lösung $\boldsymbol{\mathcal{S}}$, für die gilt dass $\boldsymbol{\mathcal{S}}\big(\mathbf{0}\big) = \boldsymbol{\mathcal{H}}^m\big(\mathbf{0}, \mathbf{0}\big)$.

3 Boltzmann-Sampler

3.1 Einleitung

Wenn wir eine kombinatorische Spezifikation betrachten, können wir uns die Frage stellen, wie wir eine zufällige Struktur der Spezies, die durch die Spezifikation definiert ist, erzeugen können. Dabei können wir auch noch fordern, dass die Wahrscheinlichkeitsverteilung gewissen Ansprüchen genügt, zum Beispiel, dass alle Strukturen einer festen Größe gleich wahrscheinlich gezogen werden. Weiters stellt sich die Frage, ob wir eine gewünschte Strukturgröße festlegen können, die gezogen werden soll.

Es gibt mehrere Lösungen dieser Fragestellung, wie zum Beispiel die *Rekursive Methode* [6]. Die hier vorgestellten *Boltzmann-Sampling-Algorithmen* wurden in [4] beziehungsweise in [5] eingeführt. Hierbei handelt es sich um eine sehr effiziente Methode, Zufallsstrukturen zu erzeugen. Wenn man die Einschränkung der Größe der gesampelten Strukturen, fallen lässt, dann erzeugen die Boltzmann-Sampler Strukturen der Größe n in $O(n)$ Zeitbedarf. Das Sampling von Strukturen einer fest vorgegebenen Größe ist immerhin noch in $O(n^2)$ möglich.

Es existieren sowohl Boltzmann-Sampler für markierte als auch für unmarkierte Strukturen. Die Sampler für markierte Strukturen werden *exponentielle* Boltzmann-Sampler genannt und werden im nächsten Abschnitt behandelt. Das Problem des Samplings unmarkierter Strukturen, also Isomorphieklassen, wird von den *gewöhnlichen* Boltzmann-Samplern gelöst. Diese werden allerdings in dieser Arbeit nicht behandelt. Eine auf der Spezies-Theorie basierende Untersuchung dieser Sampler findet man in [3, S. 28-40].

Für die Definition der Boltzmann-Sampler ist es notwendig, dass einer von einer Spezies erzeugten Struktur in eindeutiger Weise eine *Größe* zugeordnet werden kann. Unter der Größe einer Struktur verstehen wir die Kardinalität der Menge, auf der die Struktur erzeugt wurde. Wenn wir nun aber die Spezies betrachten, die auf jeder beliebigen endlichen Menge genau die leere Menge $s := \emptyset$ als Struktur erzeugt, so kann dieser Struktur s keine Größe in diesem Sinn zugeordnet werden.

Wir können jede Spezies $\mathcal{F}$ dahingehend erweitern, dass für alle endlichen Mengen U_1 und U_2 gilt, dass $\mathcal{F}[U_1] \cap \mathcal{F}[U_2] = \emptyset$ ist.

Definition 3.1.1. Sei $\mathcal{F}$ eine kombinatorische Spezies, dann ist die *durch die Größe erweiterte Spezies* $\mathcal{F}$, auch als $\mathcal{F}_g$ bezeichnet, jene Spezies, die auf einer beliebigen endlichen Menge U die Strukturen $\mathcal{F}[U] \times \{U\}$ erzeugt, und deren Transport der Strukturen für beliebige endliche Mengen U_1, U_2 und eine beliebige Bijektion $\sigma : U_1 \to U_2$ als $\mathcal{F}_g[\sigma]\big((s, U_1)\big) := \big(\mathcal{F}[\sigma](s), U_2\big)$ für alle $(s, U_1) \in \mathcal{F}_g[U_1]$ definiert ist.

Sei weiters $\tilde{s} := (s, U) \in \mathcal{F}_g[U]$, dann ist die *Größe* der Struktur $\tilde{s}$, auch als $|\tilde{s}|$ bezeichnet, die Kardinalität der Menge U.

In Zukunft können wir uns ohne weitere Adaptionen jede verwendete Spezies durch die Größe erweitert denken, und werden auch die eingeführten Bezeichnungen der Spezies für die jeweils erweiterte Spezies verwenden, denn es gilt der folgende Satz.

Satz 3.1.2. *Sei $\mathcal{F}$ eine beliebige kombinatorische Spezies, dann ist $\mathcal{F}$ isomorph zu $\mathcal{F}_g$.*

Beweis. Wir definieren für jede endliche Menge U die Abbildung $\alpha_U : \mathcal{F}[U] \to \mathcal{F}_g[U], s \mapsto (s, U)$. Nun gilt für alle endlichen Mengen U, V, für alle Bijektionen $\sigma : U \to V$ und alle Strukturen $s \in \mathcal{F}[U]$

$$\mathcal{F}_g[\sigma]\big(\alpha_U(s)\big) = \mathcal{F}_g[\sigma]\big((s, U)\big) = \big(\mathcal{F}[\sigma](s), V\big) = \alpha_V\big(\mathcal{F}[\sigma](s)\big).$$

Die Familie $\big(\alpha_U\big)$, mit allen endlichen Mengen indiziert, ist also ein Isomorphismus von $\mathcal{F}$ nach $\mathcal{F}_g$. □

Für die Definition eines Boltzmann-Samplers einer Spezies ist es notwendig, dass die dem Typ des Samplers entsprechende erzeugende Funktion eine an der Stelle 0 analytische Funktion repräsentiert. Daher untersuchen wir noch, welche Bedingungen von kombinatorische Systemen erfüllt werden müssen, damit sie für das Boltzmann-Sampling geeignet sind.

Definition 3.1.3. Eine *konstruierbare Spezies* ist entweder

1. eine der *Basis-Spezies* aus der Menge $\{\mathbf{1}, \mathcal{Z}, +, \cdot, \textsc{Seq}, \textsc{Cyc}, \textsc{Set}, \mathcal{Y}_1, \mathcal{Y}_2, \ldots\}$,
2. eine Basis-Spezies mit Einschränkung der Kardinalität auf eine endliche Vereinigung von Intervallen,
3. die Komposition von konstruierbaren Spezies, oder
4. die Lösung eines wohlfundierten Systems $\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{H}}\big(\mathcal{Z}, \boldsymbol{\mathcal{Y}}\big)$, sodass jede Koordinate von $\boldsymbol{\mathcal{H}}$ konstruierbar ist.

Eine Spezies, die nur durch die ersten 3 Regeln definiert ist (also ohne Rekursivität), nennen wir *iterativ konstruierbar*.

Definition 3.1.4. Eine Spezies $\boldsymbol{\mathcal{H}}\big(\boldsymbol{\mathcal{Z}}\big)$ heißt *analytisch* genau dann, wenn die exponentielle erzeugende Funktion $\mathbf{H}(\mathbf{z})$ auf einer Nullumgebung analytisch ist.

Nun gilt folgender Satz, der in [14, S. 42] bewiesen ist.

Satz 3.1.5. *Sei $\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{H}}\big(\mathcal{Z}, \boldsymbol{\mathcal{Y}}\big)$ ein wohlfundiertes System, wobei $\boldsymbol{\mathcal{H}}$ ein Vektor von konstruierbaren Spezies ist, dann ist die Lösung dieses Systems eine Spezies, sodass für exponentielle und gewöhnliche erzeugende Funktion jeweils Nullumgebungen existieren, auf denen sie analytisch sind.*

Aufgrund dieses Satzes und der Praxisrelevanz der konstruierbaren Spezies werden im nächsten Abschnitt Sampler für die angegebenen Basis-Spezies und die Komposition von Spezies angegeben und analysiert.

3.2 Exponentielle Boltzmann-Sampler

In diesem Abschnitt betrachten wir die exponentiellen Boltzmann-Sampler. Diese erzeugen markierte Zufallsstrukturen. Die Definitionen und Beweise, die in [5, S. 593-597] angegeben sind, werden im Kontext der Speziestheorie formuliert, genauer ausgeführt und verallgemeinert.
Zuerst betrachten wir die Wahrscheinlichkeitsverteilung, mit der der Sampler Strukturen erzeugt. Diese ist durch die folgende Klasse von Wahrscheinlichkeitsräumen gegeben.

Definition 3.2.1. Sei $\mathcal{F}$ eine analytische kombinatorische Spezies, $F(z) = \sum_{n\geq 0} \frac{f_n}{n!} z^n$ deren exponentielle erzeugende Funktion und $\rho > 0$ der Konvergenzradius der durch $F(z)$ induzierten Potenzreihe. Sei nun $x \in (0, \rho)$ und $\big(U_n\big)_{n\in\mathbb{N}_0}$ eine Folge von endlichen Mengen mit $|U_n| = n$ für alle $n \in \mathbb{N}_0$, dann wird der Wahrscheinlichkeitsraum $\big(\Omega, \mathbf{2}^\Omega, P\big)$ mit $\Omega := \bigcup_{n\in\mathbb{N}_0} \mathcal{F}[U_n]$ genau dann *exponentielles Boltzmann-Modell mit Parameter x* bezüglich der Mengenfolge $\big(U_n\big)_{n\in\mathbb{N}_0}$ genannt, wenn für das Wahrscheinlichkeitsmaß P gilt, dass

$$P\big(\{s\}\big) = \frac{x^{|s|}}{|s|!F(x)} \text{ für alle } s \in \Omega.$$

Dass es sich hierbei um einen Wahrscheinlichkeitsraum handelt, sehen wir, weil gilt, dass

$$\begin{aligned} P\big(\Omega\big) &= P\Big(\bigcup_{n\in\mathbb{N}_0} \mathcal{F}[U_n]\Big) = \sum_{n\in\mathbb{N}_0} P\big(\mathcal{F}[U_n]\big) = \sum_{n\in\mathbb{N}_0} P\Big(\bigcup_{s\in\mathcal{F}[U_n]} \{s\}\Big) = \sum_{n\in\mathbb{N}_0} \sum_{s\in\mathcal{F}[U_n]} P\big(\{s\}\big) \\ &= \sum_{n\in\mathbb{N}_0} \sum_{s\in\mathcal{F}[U_n]} \frac{x^{|s|}}{|s|!F(x)} = \sum_{n\in\mathbb{N}_0} \sum_{s\in\mathcal{F}[U_n]} \frac{x^n}{n!F(x)} = \sum_{n\in\mathbb{N}_0} \frac{f_n x^n}{n!F(x)} \\ &= \frac{1}{F(x)} \sum_{n\in\mathbb{N}_0} \frac{f_n}{n!} x^n = \frac{F(x)}{F(x)} = 1. \end{aligned}$$

Weiters sehen wir, dass die Strukturen einer festen Größe alle gleich wahrscheinlich sind, denn es gilt für alle $n \in \mathbb{N}_0$

$$P\big(\mathcal{F}[U_n]\big) = \bigcup_{s\in\mathcal{F}[U_n]} \{s\} (=) \sum_{s\in\mathcal{F}[U_n]} P\big(\{s\}\big) = \sum_{s\in\mathcal{F}[U_n]} \frac{x^n}{n!F(x)} = \frac{f_n x^n}{n!F(x)},$$

und für alle $n \in \mathbb{N}_0$ mit $\mathcal{F}[U_n] \neq \emptyset$ und alle $s \in \mathcal{F}[U_n]$

$$P\big(\{s\}\big|\mathcal{F}[U_n]\big) = \frac{P\big(\{s\} \cap \mathcal{F}[U_n]\big)}{P\big(\mathcal{F}[U_n]\big)} = \frac{P\big(\{s\}\big)}{P\big(\mathcal{F}[U_n]\big)} = \frac{x^n}{n!F(x)} \cdot \frac{n!F(x)}{f_n x^n} = \frac{1}{f_n}.$$

Ein exponentieller Sampler ist nun ein Sampler, der Strukturen gemäß dem exponentiellen Boltzmann-Modell erzeugt.

Definition 3.2.2. Sei $\mathcal{F}$ eine analytische kombinatorische Spezies, $F(z)$ deren exponentielle erzeugende Funktion und $\rho > 0$ der Konvergenzradius der durch $F(z)$ induzierten Potenzreihe. Sei nun $x \in (0, \rho)$ und $\big(U_n\big)_{n\in\mathbb{N}_0}$ eine Folge von endlichen Mengen mit $|U_n| = n$ für alle $n \in \mathbb{N}_0$, dann ist ein *exponentieller Boltzmann-Sampler* für markierte Strukturen ein Zufallsgenerator $\Gamma\mathcal{F}\big(x, \big(U_n\big)_{n\in\mathbb{N}_0}\big)$, der für die Spezies $\mathcal{F}$ das Boltzmann-Modell mit Parameter x bezüglich der Mengenfolge $\big(U_n\big)_{n\in\mathbb{N}_0}$ realisiert. Wenn aus dem Kontext klar ist, um welchen Boltzmann-Parameter und um welche Mengenfolge es sich handelt, werden wir auch $\Gamma\mathcal{F}$ schreiben.

Zur Erzeugung von Strukturen der Mengen-, Folgen- und Zyklen-Spezies benötigen wir Generatoren für nach den folgenden Wahrscheinlichkeitsverteilungen verteilten Zufallszahlen. $(\Omega, \mathbf{2}^\Omega, P)$ bezeichnet in den folgenden Definitionen einen beliebigen Wahrscheinlichkeitsraum.

Definition 3.2.3. Sei $p \in (0, 1]$, dann ist eine Zufallsvariable $X : (\Omega, \mathbf{2}^\Omega, P) \to \mathbb{N}_0$ genau dann *geometrisch* mit Parameter p verteilt, wenn für alle $n \in \mathbb{N}_0$ gilt, dass $P\big(X = n\big) = (1-p)^n p$. Ein Generator einer derartigen Zufallsvariable wird in Folge mit Geom(p) bezeichnet.

Definition 3.2.4. Sei $p \geq 0$, dann ist eine Zufallsvariable $X : (\Omega, \mathbf{2}^\Omega, P) \to \mathbb{N}_0$ genau dann mit Parameter p *Poisson-verteilt*, wenn für alle $n \in \mathbb{N}_0$ gilt, dass $P\big(X = n\big) = \exp(-p)\frac{p^n}{n!}$. Ein Generator einer derartigen Zufallsvariable wird in Folge mit Pois(p) bezeichnet.

Definition 3.2.5. Sei $p \in (0, 1]$, dann ist eine Zufallsvariable $X : (\Omega, \mathbf{2}^\Omega, P) \to \mathbb{N}$ genau dann mit Parameter p *logarithmisch verteilt*, wenn für alle $n \in \mathbb{N}$ gilt, dass $P\big(X = n\big) = -\frac{p^n}{n\log(1-p)}$. Ein Generator einer derartigen Zufallsvariable wird in Folge mit Loga(p) bezeichnet.

Weiters werden wir in Folge einen Generator, der eine auf $[0, 1)$ gleichverteilte Zufallsvariable erzeugt, mit Uniform bezeichnen.

Definition 3.2.6. Seien A, B endliche Mengen mit $|A| = |B| =: n$, dann bezeichnet Bijection(A, B) einen Generator, der gleichverteilt eine zufällige Bijektion zwischen A und B erzeugt. Für jede Bijektion $\pi : A \to B$ gilt hierbei also $P\big(\text{Bijection}(A, B) = \pi\big) = \frac{1}{n!}$.

Definition 3.2.7. Seien A, B Mengen mit $|A| = |B|$, dann bezeichnet $\text{Bij}\big(A, B\big)$ die Menge der Bijektionen zwischen A und B, also $\text{Bij}\big(A, B\big) = \{f : A \to B : f \text{ bijektiv}\}$.

Nun wenden wir uns konkreten Boltzmann-Samplern zu. Zuerst werden wir die Sampler für die Spezies $\mathbf{1}$, $\mathcal{Z}$, SEQ, SET und CYC betracheten, dann werden wir sehen, wie wir Boltzmann-Sampler für die eingeführten Operationen auf Spezies erzeugen können.

Der erste Sampler, den wir betrachten, erzeugt eine zufällige Struktur der $\mathbf{1}$-Spezies, die nach dem exponentiellen Boltzmann-Modell verteilt ist.

Satz 3.2.8. *Algorithmus 1 ist ein exponentieller Boltzmann-Sampler der* $\mathbf{1}$*-Spezies.*

Algorithmus 1 $\Gamma\mathbf{1}\big(x, (U_n)_{n\in\mathbb{N}_0}\big)$, exponentieller Boltzmann-Sampler der **1**-Spezies

Eingabe: Boltzmann-Parameter $x > 0$ und Mengenfolge $(U_n)_{n\in\mathbb{N}_0}$ mit $|U_n| = n$ für alle $n \in \mathbb{N}_0$.
Ausgabe: Struktur $s \in \bigcup_{n\in\mathbb{N}_0} \mathbf{1}[U_n]$, die nach dem exponentiellen Boltzmann-Modell verteilt ist.
return $\emptyset$

Algorithmus 2 $\Gamma\mathcal{Z}\big(x, (U_n)_{n\in\mathbb{N}_0}\big)$, exponentieller Boltzmann-Sampler der $\mathcal{Z}$-Spezies

Eingabe: Boltzmann-Parameter $x > 0$ und Mengenfolge $(U_n)_{n\in\mathbb{N}_0}$ mit $|U_n| = n$ für alle $n \in \mathbb{N}_0$.
Ausgabe: Struktur $s \in \bigcup_{n\in\mathbb{N}_0} \mathcal{Z}[U_n]$, die nach dem exponentiellen Boltzmann-Modell verteilt ist.
return U_1

Beweis. Das exponentielle Boltzmann-Modell sieht im Fall der **1**-Spezies derart aus, dass der Merkmalsraum $\Omega = \bigcup_{n\in\mathbb{N}_0} \mathbf{1}[U_n] = \{\emptyset\}$ genau aus der leeren Menge besteht. Es gilt daher $P\big(\{\emptyset\}\big) = \frac{x^{|\emptyset|}}{|\emptyset|!\mathbf{1}(x)} = \frac{x^0}{0!\cdot 1} = 1$. Der angegebene Algorithmus gibt die leere Menge mit Wahrscheinlichkeit 1 zurück. □

Satz 3.2.9. *Algorithmus 2 ist ein exponentieller Boltzmann-Sampler der $\mathcal{Z}$-Spezies.*

Beweis. Im Fall der $\mathcal{Z}$-Spezies besitzt das exponentielle Boltzmann-Modell den Merkmalsraum $\Omega = \bigcup_{n\in\mathbb{N}_0} \mathcal{Z}[U_n] = \{U_1\}$. Die Wahrscheinlichkeit des einzigen Elements des Merkmalsraum ist $P\big(\{U_1\}\big) = \frac{x^{|U_1|}}{|U_1|!Z(x)} = \frac{x^1}{1!\cdot x} = 1$. Der angegebene Algorithmus gibt U_1 mit Wahrscheinlichkeit 1 zurück. □

Algorithmus 3 $\Gamma\textsc{Seq}\big(x, (U_n)_{n\in\mathbb{N}_0}\big)$, exponentieller Boltzmann-Sampler der SEQ-Spezies

Eingabe: Boltzmann-Parameter $x \in (0, 1)$ und Mengenfolge $(U_n)_{n\in\mathbb{N}_0}$ mit $|U_n| = n$ für alle $n \in \mathbb{N}_0$.
Ausgabe: Struktur $s \in \bigcup_{n\in\mathbb{N}_0} \textsc{Seq}[U_n]$, die nach dem exponentiellen Boltzmann-Modell verteilt ist.

```
k ← Geom(1 − x)
if k = 0 then
    return ∅
else
    π ← Bijection([k], U_k)
    return (π(1), π(2), ..., π(k))
end if
```

Satz 3.2.10. *Algorithmus 3 ist ein exponentieller Boltzmann-Sampler der* SEQ*-Spezies.*

Beweis. Wir sehen leicht, dass die Größe der der zurückgegebenen Struktur dem Wert der Variable k entspricht. Es gilt daher

$$P\big(|\Gamma\textsc{Seq}| = n\big) = P\big(k = n\big) = \big(1 - (1 - x)\big)^n \cdot (1 - x) = x^n \cdot (1 - x).$$

Für den Fall, dass die Strukturgröße 0 ist, wird die einzige Struktur dieser Größe $\emptyset$ mit Wahrscheinlichkeit $1 = \frac{1}{0!}$ zurückgegeben. Für die Strukturgrößen $k > 0$ wird, da alle Bijektionen gleich wahrscheinlich erzeugt werden, jede Folge mit Wahrscheinlichkeit $\frac{1}{k!}$ zurückgegeben. Es gilt also

$$P\big(\Gamma\textsc{Seq} = s \big| |\Gamma\textsc{Seq}| = |s|\big) = \frac{1}{n!} \text{ für alle } s \in \bigcup_{n\in\mathbb{N}_0} \textsc{Seq}[U_n].$$

Insgesamt gilt also für alle Strukturen $s \in \bigcup_{n\in\mathbb{N}_0} \textsc{Seq}[U_n]$, dass

$$P\big(\Gamma\textsc{Seq} = s\big) = P\big(\Gamma\textsc{Seq} = s \big| |\Gamma\textsc{Seq}| = |s|\big) \cdot P\big(|\Gamma\textsc{Seq}| = |s|\big) = \frac{1}{|s|!} \cdot x^{|s|} \cdot (1-x).$$
$$= \frac{x^{|s|}}{|s|! \cdot \frac{1}{1-x}} = \frac{x^{|s|}}{|s|! \cdot \textsc{Seq}(x)}.$$

Der angegebene Sampler realisiert daher das exponentielle Boltzmann-Modell für die Spezies der endlichen Folgen. □

Algorithmus 4 $\Gamma\textsc{Set}\big(x, \big(U_n\big)_{n\in\mathbb{N}_0}\big)$, exponentieller Boltzmann-Sampler der Set-Spezies

Eingabe: Boltzmann-Parameter $x > 0$ und Mengenfolge $\big(U_n\big)_{n\in\mathbb{N}_0}$ mit $|U_n| = n$ für alle $n \in \mathbb{N}_0$.
Ausgabe: Struktur $s \in \bigcup_{n\in\mathbb{N}_0} \textsc{Set}[U_n]$, die nach dem exponentiellen Boltzmann-Modell verteilt ist.
$k \leftarrow \text{Pois}(x)$
return U_k

Satz 3.2.11. *Algorithmus 4 ist ein exponentieller Boltzmann-Sampler der* Set*-Spezies.*

Beweis. Zu jeder beliebigen Größe n existiert genau eine Set-Struktur dieser Größe, nämlich U_n. Wie wir leicht sehen, gilt

$$P\big(\Gamma\textsc{Set} = U_n\big) = P\big(k = n\big) = \exp(-x)\frac{x^n}{n!} = \frac{x^n}{n!\exp(x)} = \frac{x^n}{n! \cdot \textsc{Set}(x)}$$

für alle $n \in \mathbb{N}_0$. Der Rückgabewert ist daher nach dem exponentiellen Boltzmann-Modell verteilt. □

Algorithmus 5 $\Gamma\textsc{Cyc}\big(x, \big(U_n\big)_{n\in\mathbb{N}_0}\big)$, exponentieller Boltzmann-Sampler der Cyc-Spezies

Eingabe: Boltzmann-Parameter $x \in (0,1)$ und Mengenfolge $(U_n)_{n\in\mathbb{N}_0}$ mit $|U_n| = n$ für alle $n \in \mathbb{N}_0$.
Ausgabe: Struktur $s \in \bigcup_{n\in\mathbb{N}_0} \textsc{Cyc}[U_n]$, die nach dem exponentiellen Boltzmann-Modell verteilt ist.
$k \leftarrow \text{Loga}(x)$
$\pi \leftarrow \text{Bijection}([k], U_k)$
return Zyklus $\big(\pi(1)\pi(2)\dots\pi(k)\big)$

Satz 3.2.12. *Algorithmus 5 ist ein exponentieller Boltzmann-Sampler der* Cyc*-Spezies.*

Beweis. Die Größe der zurückgegebenen Struktur entspricht dem Wert der Variablen k. Es gilt also für alle $n \in \mathbb{N}$, dass

$$P\big(|\Gamma\textsc{Cyc}| = n\big) = -\frac{x^n}{n \cdot \log(1-x)} = \frac{x^n}{n \cdot \log\frac{1}{1-x}} = \frac{x^n}{n \cdot \textsc{Cyc}(x)}.$$

Sei nun $n \in \mathbb{N}$ und $s := \big(u_1 u_2 \dots u_n\big) \in \textsc{Cyc}[U_n]$, dann gibt es n Bijektionen $\pi_1, \dots, \pi_n$ zwischen $[n]$ und U_n, sodass $\big(\pi_i(1)\pi_i(2)\dots\pi_i(n)\big) = s$ für $i = 1..n$. Jede dieser Bijektionen wird mit Wahrscheinlichkeit $\frac{1}{n!}$ der Variable π zugewiesen. Wir sehen daher, dass

$$P\big(\Gamma\textsc{Cyc} = s \big| |\Gamma\textsc{Cyc}| = |s|\big) = \frac{|s|}{|s|!} = \frac{1}{(|s|-1)!}.$$

Insgesamt gilt also, dass

$$\begin{aligned} P\big(\Gamma\textsc{Cyc} = s\big) &= P\big(\Gamma\textsc{Cyc} = s \big| |\Gamma\textsc{Cyc}| = |s|\big) \cdot P\big(|\Gamma\textsc{Cyc}| = |s|\big) \\ &= \frac{1}{(|s|-1)!} \cdot \frac{x^{|s|}}{|s|\,\textsc{Cyc}(x)} = \frac{x^n}{|s|! \cdot \textsc{Cyc}(x)}. \end{aligned}$$

Der Algorithmus realisiert also das exponentielle Boltzmann-Modell. □

Nun wenden wir uns der Erzeugung von Samplern, die kombinatorische Operationen realisieren zu. Wenn wir kombinatorische Spezies $\mathcal{A}$ und $\mathcal{B}$ betrachten, zu denen wir exponentielle Boltzmann-Sampler $\Gamma\mathcal{A}$ und $\Gamma\mathcal{B}$ kennen, so können wir aus diesen Samplern in automatischer Weise einen exponentiellen Boltzmann-Sampler der Summe $\mathcal{A}+\mathcal{B}$, des Produkts $\mathcal{A}\cdot\mathcal{B}$ und, falls $\mathcal{B}[\emptyset] = \emptyset$, der Substitution $\mathcal{A}\circ\mathcal{B}$ konstruieren.

Algorithmus 6 $\Gamma(\mathcal{A}+\mathcal{B})\big(x, (U_n)_{n\in\mathbb{N}_0}\big)$, exponentieller Boltzmann-Sampler der Summe der Spezies $\mathcal{A}$ und $\mathcal{B}$

Eingabe: Exponentieller Boltzmann-Sampler $\Gamma\mathcal{A}$ von $\mathcal{A}$, exponentieller Boltzmann-Sampler $\Gamma\mathcal{B}$ von $\mathcal{B}$, Boltzmann-Parameter $x \in \big(0, \min(\rho_A, \rho_B)\big)$, wobei ρ_A den Konvergenzradius der exponentiellen erzeugenden Funktion $A(z)$ von $\mathcal{A}$ und ρ_B den Konvergenzradius der exponentiellen erzeugenden Funktion $B(z)$ von $\mathcal{B}$ bezeichnet, und eine Mengenfolge $(U_n)_{n\in\mathbb{N}_0}$ mit $|U_n| = n$ für alle $n \in \mathbb{N}_0$.

Ausgabe: Struktur $s \in \bigcup_{n\in\mathbb{N}_0}(\mathcal{A}+\mathcal{B})[U_n]$, die nach dem exponentiellen Boltzmann-Modell verteilt ist.

$\lambda \leftarrow$ Uniform()
if $\lambda < \frac{A(x)}{A(x)+B(x)}$ **then**
  return $\big(\Gamma\mathcal{A}\big(x, (U_n)_{n\in\mathbb{N}_0}\big), 1\big)$
else
  return $\big(\Gamma\mathcal{B}\big(x, (U_n)_{n\in\mathbb{N}_0}\big), 2\big)$
end if

Satz 3.2.13. *Seien $\mathcal{A}$ und $\mathcal{B}$ zwei kombinatorische Spezies, zu denen exponentielle Boltzmann-Sampler $\Gamma\mathcal{A}$ und $\Gamma\mathcal{B}$ existieren, so ist Algorithmus 6 ein exponentieller Boltzmann-Sampler der Summenspezies $\mathcal{A}+\mathcal{B}$.*

Beweis. Zuerst berechnen wir die Wahrscheinlichkeit, dass eine $\mathcal{A}$- beziehungsweise eine $\mathcal{B}$-Struktur zurückgegeben wird. Es gilt

$$\begin{aligned} P\big(\Gamma(\mathcal{A}+\mathcal{B}) \in \bigcup_{n\in\mathbb{N}_0} \mathcal{A}[U_n]\times\{1\}\big) &= P\left(\lambda < \frac{A(x)}{A(x)+B(x)}\right) = \frac{A(x)}{A(x)+B(x)}, \text{ und} \\ P\big(\Gamma(\mathcal{A}+\mathcal{B}) \in \bigcup_{n\in\mathbb{N}_0} \mathcal{B}[U_n]\times\{2\}\big) &= P\left(\lambda \geq \frac{A(x)}{A(x)+B(x)}\right) = 1 - P\left(\lambda < \frac{A(x)}{A(x)+B(x)}\right) \\ &= 1 - \frac{A(x)}{A(x)+B(x)} = \frac{A(x)+B(x)-A(x)}{A(x)+B(x)} = \frac{B(x)}{A(x)+B(x)}. \end{aligned}$$

Nun betrachten wir die Wahrscheinlichkeit, dass eine konkrete $\mathcal{A}$- bzw. $\mathcal{B}$-Struktur zurückgegeben

wird. Sei $s \in \big(\bigcup_{n\in\mathbb{N}_0} \mathcal{A}[U_n] \times \{1\}\big) \cup \big(\bigcup_{n\in\mathbb{N}_0} \mathcal{B}[U_n] \times \{2\}\big)$, dann gilt

$$P\Bigg(\Gamma(\mathcal{A}+\mathcal{B})\Bigg|\Gamma(\mathcal{A}+\mathcal{B}) \in \bigcup_{n\in\mathbb{N}_0} \mathcal{A}[U_n] \times \{1\}\Bigg) = \begin{cases} P\big(\Gamma\mathcal{A} = s\big) & \text{falls } s \in \bigcup_{n\in\mathbb{N}_0} \mathcal{A}[U_n] \times \{1\}, \\ 0 & \text{sonst,} \end{cases}$$

$$= \begin{cases} \frac{x^{|s|}}{|s|!\cdot A(x)} & \text{falls } s \in \bigcup_{n\in\mathbb{N}_0} \mathcal{A}[U_n] \times \{1\}, \\ 0 & \text{sonst,} \end{cases} = \mathbb{1}_{\bigcup_{n\in\mathbb{N}_0} \mathcal{A}[U_n]\times\{1\}}(s) \cdot \frac{x^{|s|}}{|s|! \cdot A(x)}, \text{ und}$$

$$P\Bigg(\Gamma(\mathcal{A}+\mathcal{B})\Bigg|\Gamma(\mathcal{A}+\mathcal{B}) \in \bigcup_{n\in\mathbb{N}_0} \mathcal{B}[U_n] \times \{2\}\Bigg) = \begin{cases} P\big(\Gamma\mathcal{B} = s\big) & \text{falls } s \in \bigcup_{n\in\mathbb{N}_0} \mathcal{B}[U_n] \times \{2\}, \\ 0 & \text{sonst,} \end{cases}$$

$$= \begin{cases} \frac{x^{|s|}}{|s|!\cdot B(x)} & \text{falls } s \in \bigcup_{n\in\mathbb{N}_0} \mathcal{B}[U_n] \times \{2\}, \\ 0 & \text{sonst,} \end{cases} = \mathbb{1}_{\bigcup_{n\in\mathbb{N}_0} \mathcal{B}[U_n]\times\{2\}}(s) \cdot \frac{x^{|s|}}{|s|! \cdot B(x)}.$$

Die Wahrscheinlichkeit, dass der Algorithmus die Struktur s zurückgibt, ist daher

$$\begin{aligned} P\big(\Gamma(\mathcal{A}+\mathcal{B}) = s\big) = {} & P\Bigg(\Gamma(\mathcal{A}+\mathcal{B}) = s\Bigg|\Gamma(\mathcal{A}+\mathcal{B}) \in \bigcup_{n\in\mathbb{N}_0} \mathcal{A}[U_n] \times \{1\}\Bigg) \cdot \\ & P\Bigg(\Gamma(\mathcal{A}+\mathcal{B}) \in \bigcup_{n\in\mathbb{N}_0} \mathcal{A}[U_n] \times \{1\}\Bigg) + \\ & P\Bigg(\Gamma(\mathcal{A}+\mathcal{B}) = s\Bigg|\Gamma(\mathcal{A}+\mathcal{B}) \in \bigcup_{n\in\mathbb{N}_0} \mathcal{B}[U_n] \times \{2\}\Bigg) \cdot \\ & P\Bigg(\Gamma(\mathcal{A}+\mathcal{B}) \in \bigcup_{n\in\mathbb{N}_0} \mathcal{B}[U_n] \times \{2\}\Bigg) \\ = {} & \mathbb{1}_{\bigcup_{n\in\mathbb{N}_0} \mathcal{A}[U_n]\times\{1\}}(s) \cdot \frac{x^{|s|}}{|s|! \cdot A(x)} \cdot \frac{A(x)}{A(x)+B(x)} + \\ & \mathbb{1}_{\bigcup_{n\in\mathbb{N}_0} \mathcal{B}[U_n]\times\{2\}}(s) \cdot \frac{x^{|s|}}{|s|! \cdot B(x)} \cdot \frac{B(x)}{A(x)+B(x)} \\ = {} & \underbrace{\big(\mathbb{1}_{\bigcup_{n\in\mathbb{N}_0} \mathcal{A}[U_n]\times\{1\}}(s) + \mathbb{1}_{\bigcup_{n\in\mathbb{N}_0} \mathcal{B}[U_n]\times\{2\}}(s)\big)}_{=1} \cdot \frac{x^{|s|}}{|s|!(A(x)+B(x))}. \end{aligned}$$

Wir sehen also, dass der angegebene Algorithmus ein exponentieller Boltzmann-Sampler der Summe von $\mathcal{A}$ und $\mathcal{B}$ ist. □

Algorithmus 7 $\Gamma(\mathcal{A}\cdot\mathcal{B})\big(x, (U_n)_{n\in\mathbb{N}_0}\big)$, exponentieller Boltzmann-Sampler des Produkts der Spezies $\mathcal{A}$ und $\mathcal{B}$

Eingabe: Exponentieller Boltzmann-Sampler $\Gamma\mathcal{A}$ von $\mathcal{A}$, exponentieller Boltzmann-Sampler $\Gamma\mathcal{B}$ von $\mathcal{B}$, Boltzmann-Parameter $x \in \big(0, \min(\rho_A, \rho_B)\big)$, wobei ρ_A den Konvergenzradius der exponentiellen erzeugenden Funktion $A(z)$ von $\mathcal{A}$ und ρ_B den Konvergenzradius der exponentiellen erzeugenden Funktion $B(z)$ von $\mathcal{B}$ bezeichnet, und eine Mengenfolge $(U_n)_{n\in\mathbb{N}_0}$ mit $|U_n| = n$ für alle $n \in \mathbb{N}_0$.

Ausgabe: Struktur $s \in \bigcup_{n\in\mathbb{N}_0}(\mathcal{A}\cdot\mathcal{B})[U_n]$, die nach dem exponentiellen Boltzmann-Modell verteilt ist.

$a \leftarrow \Gamma\mathcal{A}\big(x, ([n]\times\{1\})_{n\in\mathbb{N}_0}\big)$

$b \leftarrow \Gamma\mathcal{B}\big(x, ([n]\times\{2\})_{n\in\mathbb{N}_0}\big)$

$\pi \leftarrow \text{Bijection}\big(([|a|]\times\{1\}) \cup ([|b|]\times\{2\}), U_{|a|+|b|}\big)$

return $\Big(\mathcal{A}\big[\pi\big|_{[|a|]\times\{1\}}\big](a), \mathcal{B}\big[\pi\big|_{[|b|]\times\{2\}}\big](b), \pi\big([|a|]\times\{1\}\big), \pi\big([|b|]\times\{2\}\big)\Big)$

Satz 3.2.14. *Seien $\mathcal{A}$ und $\mathcal{B}$ zwei kombinatorische Spezies, zu denen exponentielle Boltzmann-Sampler $\Gamma\mathcal{A}$ und $\Gamma\mathcal{B}$ existieren, so ist der Algorithmus γ ein exponentieller Boltzmann-Sampler der Produktspezies $\mathcal{A}\cdot\mathcal{B}$.*

Beweis. Es ist von Vorteil, den Wahrscheinlichkeitsraum zu betrachten, dessen Element das Tupel (a,b,π) vor der Transformation im Rückgabeschritt ist. Für den Merkmalsraum dieses Wahrscheinlichkeitsraumes Ω gilt

$$\Omega = \bigcup_{k,l\geq 0} \mathcal{A}\big[[k]\times\{1\}\big]\times\mathcal{B}\big[[l]\times\{2\}\big]\times \mathrm{Bij}\big(([k]\times\{1\})\cup([l]\times\{2\}), U_{k+l}\big).$$

Ausserdem sehen wir leicht, dass aufgrund der unabhängigen Aufrufe der Teilsampler $\Gamma\mathcal{A}$ und $\Gamma\mathcal{B}$ und der Gleichverteilung der ermittelten Bijektion für alle $(a,b,\pi)\in\Omega$

$$P\big(\{(a,b,\pi)\}\big) = \frac{x^{|a|}}{|a|!\cdot A(x)}\cdot\frac{x^{|b|}}{|b|!\cdot B(x)}\cdot\frac{1}{(|a|+|b|)!} \text{ gilt.}$$

Sei nun

$$X:\ \Omega\to\bigcup_{n\in\mathbb{N}_0}(\mathcal{A}\cdot\mathcal{B})[U_n],$$
$$(a,b,\pi)\mapsto\Big(\mathcal{A}\Big[\pi\big|_{[|a|]\times\{1\}}\Big](a),\mathcal{B}\Big[\pi\big|_{[|b|]\times\{2\}}\Big](b),\pi\big([|a|]\times\{1\}\big),\pi\big([|b|]\times\{2\}\big)\Big),$$

so gilt, dass vom Algorithmus $X\big((a,b,\pi)\big)$ zurückgegeben wird.

Betrachten wir ein beliebiges Element $(u,v,M_1,M_2)\in\bigcup_{n\in\mathbb{N}_0}(\mathcal{A}\cdot\mathcal{B})[U_n]$, so wird dieses mit Wahrscheinlichkeit $P\big(X^{-1}\big(\{(u,v,M_1,M_2)\}\big)\big)$ retourniert. Für ein beliebiges Element $(a,b,\pi)\in X^{-1}\big(\{(u,v,M_1,M_2)\}\big)$ muss gelten, dass $|a|=|u|$ und $|b|=|v|$. Also folgt, dass

$$X^{-1}\big(\{(u,v,M_1,M_2)\}\big)\subseteq\mathcal{A}\big[[|u|]\times\{1\}\big]\times\mathcal{B}\big[[|v|]\times\{2\}\big]\times\mathrm{Bij}\big(([|u|]\times\{1\})\cup([|v|]\times\{2\}),U_{|u|+|v|}\big).$$

Außerdem muss $\pi\big([|u|]\times\{1\}\big)=M_1$ und $\pi\big([|v|]\times\{2\}\big)=M_2$ sein, und wir sehen, dass wegen der Eigenschaften des Transports der Strukturen

$$\mathcal{A}\Big[\pi\big|_{[|u|]\times\{1\}}\Big](a)=u\Leftrightarrow\mathcal{A}\Big[\Big(\pi\big|_{[|u|]\times\{1\}}\Big)^{-1}\Big](u)=a,\ \text{und}$$
$$\mathcal{B}\Big[\pi\big|_{[|v|]\times\{2\}}\Big](b)=v\Leftrightarrow\mathcal{B}\Big[\Big(\pi\big|_{[|v|]\times\{2\}}\Big)^{-1}\Big](v)=b \text{ gilt.}$$

Wir können daher $X^{-1}\big(\{(u,v,M_1,M_2)\}\big)$ als

$$\Big\{\Big(\mathcal{A}\Big[\Big(\pi\big|_{[|u|]\times\{1\}}\Big)^{-1}\Big](u),\mathcal{B}\Big[\Big(\pi\big|_{[|v|]\times\{2\}}\Big)^{-1}\Big](v),\pi\Big)\ :\ \pi\in B\wedge\pi\big([|u|]\times\{1\}\big)=M_1\wedge\pi\big([|v|]\times\{2\}\big)=M_2\Big\}$$

anschreiben, wobei $B:=\mathrm{Bij}\big(([|u|]\times\{1\})\cup([|v|]\times\{2\}),U_{|u|+|v|}\big)$. Es gibt genau $|u|!\cdot|v|!$ Bijektionen aus B, die die Bedingungen $\pi\big([|u|]\times\{1\}\big)=M_1$ und $\pi\big([|v|]\times\{2\}\big)=M_2$ erfüllen. Wenn wir noch berücksichtigen, dass jedes Element aus $X^{-1}\big(\{(u,v,M_1,M_2)\}\big)$ die selbe Wahrscheinlichkeit

$$\frac{x^{|u|}}{|u|!\cdot A(x)}\cdot\frac{x^{|v|}}{|v|!\cdot B(x)}\cdot\frac{1}{(|u|+|v|)!}$$

besitzt, erhalten wir also insgesamt

$$\begin{aligned}P\big(\Gamma(\mathcal{A}\cdot\mathcal{B})=(u,v,M_1,M_2)\big)&=\big|X^{-1}\big(\{(u,v,M_1,M_2)\}\big)\big|\cdot\frac{x^{|u|}}{|u|!\cdot A(x)}\cdot\frac{x^{|v|}}{|v|!\cdot B(x)}\cdot\frac{1}{(|u|+|v|)!}\\&=\frac{x^{|u|}}{|u|!\cdot A(x)}\cdot\frac{x^{|v|}}{|v|!\cdot B(x)}\cdot\frac{|u|!\cdot|v|!}{(|u|+|v|)!}\\&=\frac{x^{|u|+|v|}}{(|u|+|v|)!\cdot A(x)B(x)}.\end{aligned}$$

Wir sehen also, dass der Algorithmus das Gewünschte leistet. □

Algorithmus 8 $\Gamma(\mathcal{A}\circ\mathcal{B})\big(x,(U_n)_{n\in\mathbb{N}_0}\big)$, exponentieller Boltzmann-Sampler der Komposition von $\mathcal{A}$ mit $\mathcal{B}$

Eingabe: Exponentieller Boltzmann-Sampler $\Gamma\mathcal{A}$ von $\mathcal{A}$, exponentieller Bolzmann-Sampler $\Gamma\mathcal{B}$ von $\mathcal{B}$ mit $\mathcal{B}[\emptyset]=\emptyset$, Boltzmann-Parameter $x\in\big(0,\rho_B\big)$ mit $B(x)\in\big(0,\rho_A\big)$, wobei ρ_A den Konvergenzradius der exponentiellen erzeugenden Funktion $A(z)$ von $\mathcal{A}$ und ρ_B den Konvergenzradius der exponentiellen erzeugenden Funktion $B(z)$ von $\mathcal{B}$ bezeichnet, und eine Mengenfolge $(U_n)_{n\in\mathbb{N}_0}$ mit $|U_n|=n$ für alle $n\in\mathbb{N}_0$.

Ausgabe: Struktur $s\in\bigcup_{n\in\mathbb{N}_0}(\mathcal{A}\circ\mathcal{B})[U_n]$, die nach dem exponentiellen Boltzmann-Modell verteilt ist.

$s\leftarrow\Gamma\mathcal{A}\big(B(x),([n])_{n\in\mathbb{N}_0}\big)$
if $|s|=0$ **then**
  return $\big(\emptyset,s,f_\emptyset\big)$
else
  for $i=1\ ..|s|$ **do**
    $t_i\leftarrow\Gamma\mathcal{B}\big(x,([n]\times\{i\})_{n\in\mathbb{N}_0}\big)$
  end for
  $\pi\leftarrow\text{Bijection}\big(\bigcup_{i=1}^{|s|}[|t_i|]\times\{i\},U_\sigma\big)$ für $\sigma:=\sum_{i=1}^{|s|}|t_i|$
  return $\Big(\big\{\beta(i)\,:\,i\in[|s|]\big\},\mathcal{A}[\beta](s),\big(\mathcal{B}[\pi_{\beta^{-1}(p)}]\big)_{p\in\{\beta(i)\,:\,i\in[|s|]\}}\Big)$, wobei für alle $i\in[|s|]$ gilt, dass $\beta(i):=\pi\big([|t_i|]\times\{i\}\big)$ und $\pi_i:=\pi\big|_{[|t_i|\times\{i\}]}$.
end if

Satz 3.2.15. *Seien $\mathcal{A}$ und $\mathcal{B}$ zwei kombinatorische Spezies mit $\mathcal{B}[\emptyset]=\emptyset$, zu denen exponentielle Boltzmann-Sampler $\Gamma\mathcal{A}$ und $\Gamma\mathcal{B}$ existieren, so ist der Algorithmus 8 ein exponentieller Boltzmann-Sampler der Komposition von $\mathcal{A}$ mit $\mathcal{B}$.*

Beweis. Zuerst bemerken wir, dass die Größe der Struktur s der Anzahl der Elemente der Partition der Grundmenge U_n entspricht. Diese Partition ist genau für die Strukturen aus $(\mathcal{A}\circ\mathcal{B})[\emptyset]$ die leere Menge. Die Verzweigung für $|s|=0$ erzeugt also ausschließlich Strukturen der Größe 0 und die andere Verzweigung ausschließlich Strukturen, die größer als 0 sind.
Betrachten wir zuerst den Fall, dass $|s|>0$. Wir sehen, dass das vom Algorithmus erzeugte Tupel $\big(s,(t_i)_{i\in[|s|]},\pi\big)$ Element der Menge $\bigcup_{\substack{k\in\mathbb{N}\\(j_1,\dots,j_k)\in\mathbb{N}^n}} M(k,j_1,\dots,j_n)$ ist, wobei

$$M(k,j_1,\dots,j_k):=\mathcal{A}[k]\times\prod_{i=1}^{k}\mathcal{B}\big[[j_i]\times\{i\}\big]\times\text{Bij}\big(\bigcup_{i=1}^{k}[j_i]\times\{i\},U_{\sum_{i=1}^k j_i}\big)\text{ gilt.}$$

Weiters gilt für alle $k\in\mathbb{N}$, für alle $\big(j_1,\dots,j_k\big)\in\mathbb{N}^n$ und jedes Tupel $\big(\tilde{s},(\tilde{t}_i)_{i\in[k]},\tilde{\pi}\big)\in M(k,j_1,\dots,j_k)$ mit $\sigma:=\sum_{i=1}^k j_i$, dass

$$\begin{aligned}P\Big(\big(s,(t_i)_{i\in[|s|]},\pi\big)=\big(\tilde{s},(\tilde{t}_i)_{i\in[n]},\tilde{\pi}\big)\Big)&=\underbrace{\frac{B(x)^k}{k!\cdot A(B(x))}}_{P\left(s=\bar{s}\right)}\cdot\prod_{i=1}^{k}\underbrace{\frac{x^{j_i}}{j_i!\cdot B(x)}}_{P\left(t_i=\bar{t_i}\right)}\cdot\underbrace{\frac{1}{\big(\sum_{i=1}^k j_i\big)!}}_{P\left(\pi=\bar{\pi}\right)}\\&=\frac{x^\sigma}{k!\cdot\prod_{i=1}^k j_i!\cdot\sigma!\cdot A(B(x))}.\end{aligned}$$

Es ist also jedes Tupel aus $M(k,j_1,\dots,j_k)$ gleich wahrscheinlich. Wir definieren die Abbildung X derart, sodass für alle $\big(s,(t_i)_{i\in[|s|]},\pi\big)\in\bigcup_{\substack{k\in\mathbb{N}\\(j_1,\dots,j_k)\in\mathbb{N}^n}} M(k,j_1,\dots,j_n)$ gilt, dass

$$X\Big(\big(s,(t_i)_{i\in[|s|]},\pi\big)\Big)=\Big(\big(\beta(i)\big)_{i\in[|s|]},\mathcal{A}[\beta](s),\big(\mathcal{B}[\pi_i](t_i)\big)_{i\in[|s|]}\Big).$$

Sei nun $n \geq 1$ und $\big(\bar{P}, \bar{s}, (\bar{t}_{\bar{p}})_{\bar{p}\in\bar{P}}\big)$ eine beliebige Struktur aus $\big(\mathcal{A} \circ \mathcal{B}\big)[U_n]$, also $\bar{P} \in \textsc{Par}[U_n]$, $\bar{s} \in \mathcal{A}[\bar{P}]$ und $\bar{t}_{\bar{p}} \in \mathcal{B}[\bar{p}]$ für alle $\bar{p} \in \bar{P}$. Sei weiters $\bar{p}_1, \ldots, \bar{p}_k$ eine beliebige Aufzählung der Elemente von $\bar{P}$, dann gilt

$$\begin{aligned}
& X\Big(\big(s, (t_i)_{i\in[|s|]}, \pi\big)\Big) = \Big(\big(\bar{p}_i\big)_{i\in[k]}, \bar{s}, \big(\bar{t}_{\bar{p}_i}\big)_{i\in[k]}\Big) \\
\Leftrightarrow\ & k = |s| \ \wedge\ \mathcal{A}[\beta](s) = \bar{s} \ \wedge \\
& \forall i \in [k] : |t_i| = |p_i| \ \wedge\ \beta(i) = \bar{p}_i \ \wedge\ \mathcal{B}[\pi_i](t_i) = \bar{t}_{\bar{p}_i} \\
\Leftrightarrow\ & \big(s, (t_i)_{i\in[|s|]}, \pi\big) \in M(k, |\bar{p}_1|, \ldots, |\bar{p}_k|) \ \wedge\ s = \mathcal{A}[\beta^{-1}](\bar{s}) \ \wedge \\
& \forall i \in [k] : \beta(i) = \bar{p}_i \ \wedge\ t_i = \mathcal{B}[\pi_i^{-1}](\bar{t}_{\bar{p}_i}).
\end{aligned}$$

Es gibt genau $\prod_{i=1}^{k} |\bar{p}_i|!$ Bijektionen $\pi \in \mathrm{Bij}\big(\bigcup_{i=1}^{k} [|\bar{p}_i|] \times \{i\}, U_n\big)$ mit $\beta(i) = \pi\big([|\bar{p}_i|] \times \{i\}\big) = \bar{p}_i$, zu jeder Bijektion π genau ein $s \in \mathcal{A}[k]$ mit $s = \mathcal{A}[\beta^{-1}](\bar{s})$ und für jedes $i \in [k]$ genau ein $t_i \in \mathcal{B}\big[[|\bar{p}_i|] \times \{i\}\big]$ mit $t_i = \mathcal{B}[\pi_i^{-1}]\big(\bar{t}_{\bar{p}_i}\big)$, da es sich bei den Transporten der Strukturen um Bijektionen handelt. Also folgt, dass $\bar{M} := X^{-1}\Big(\Big\{\big((\bar{p}_i)_{i\in[k]}, \bar{s}, (\bar{t}_{\bar{p}_i})_{i\in[k]}\big)\Big\}\Big) \subseteq M(k, |\bar{p}_1|, \ldots, |\bar{p}_k|)$ und $\big|\bar{M}\big| = \prod_{i=1}^{k} |\bar{p}_i|!$. Die Wahrscheinlichkeit der Menge $\bar{M}$ beträgt daher

$$\begin{aligned}
P\big(\bar{M}\big) &= \sum_{m\in\bar{M}} P\big(\{m\}\big) = \sum_{m\in\bar{M}} \frac{x^n}{k! \cdot \prod_{i=1}^{k} |\bar{p}_i|! \cdot n! \cdot A(B(x))} \\
&= \prod_{i=1}^{k} |\bar{p}_i|! \cdot \frac{x^n}{k! \cdot \prod_{i=1}^{k} |\bar{p}_i|! \cdot n! \cdot A(B(x))} = \frac{x^n}{k! \cdot n! \cdot A(B(x))}.
\end{aligned}$$

Sei nun Y jene Abbildung, die jedes Tupel der Gestalt $\Big(\big(a_i\big)_{i\in[l]}, b, \big(c_{a_i}\big)_{i\in[l]}\Big)$ mit $l \geq 1$ und $a_i \neq a_j$ für alle $i \neq j$ auf das Tupel $\Big(A, b, \big(c_a\big)_{a\in A}\Big)$ mit $A := \{a_i : i \in [l]\}$ abbildet, die also die Reihenfolge der Aufzählung entfernt, so sehen wir, dass der Algorithmus für den Fall, dass $|s| \geq 1$ ist, den Wert $\big(Y \circ X\big)\Big(\big(s, (t_i)_{i\in[s]}, \pi\big)\Big)$ zurückgibt. Wenn wir nun noch beachten, dass genau $k!$ Aufzählungen der Menge $\bar{P}$ existieren, so folgt, dass

$$P\Big(\big(Y \circ X\big)\Big(\big(s, (t_i)_{i\in[s]}, \pi\big)\Big) = \big(\bar{P}, \bar{s}, (\bar{t}_{\bar{p}})_{\bar{p}\in\bar{P}}\big)\Big) = k! \cdot \frac{x^n}{k! \cdot n! \cdot A(B(x))} = \frac{x^n}{n! \cdot A(B(x))}.$$

Für Strukturen, die größer als 0 sind, leistet der Algorithmus also das Gewünschte. Nun betrachten wir noch den Fall, dass $|s| = 0$. Es gilt $\big(\bar{P}, \bar{s}, (\bar{t}_{\bar{p}})_{\bar{p}\in\bar{P}}\big) \in \big(\mathcal{A} \circ \mathcal{B}\big)[\emptyset]$ genau dann, wenn $\bar{P} \in \textsc{Par}[\emptyset] = \{\emptyset\}$, $\bar{s} \in \mathcal{A}[\bar{P}]$ und $\bar{t}_{\bar{p}} \in \mathcal{B}[\bar{p}]$ für alle $\bar{p} \in \bar{P}$, also genau dann, wenn $\bar{P} = \emptyset$, $\bar{s} \in \mathcal{A}[\emptyset]$ und $(\bar{t}_{\bar{p}})_{p\in\emptyset} = f_\emptyset$. Sei also $\big(\emptyset, \bar{s}, f_\emptyset\big) \in \big(\mathcal{A} \circ \mathcal{B}\big)[\emptyset]$, dann gilt

$$\begin{aligned}
P\big(\Gamma(\mathcal{A} \circ \mathcal{B}) = \big(\emptyset, \bar{s}, f_\emptyset\big)\big) = P\big(s = \bar{s}\big) &= \frac{B(x)^{|\bar{s}|}}{|\bar{s}|! \cdot A(B(x))} \\
&= \frac{B(x)^0}{0! \cdot A(B(x))} = \frac{x^0}{0! \cdot A(B(x))}.
\end{aligned}$$

Daher leistet der Algorithmus auch für Strukturen der Größe 0 das Gewünschte. □

Bemerkung. Es ist möglich, den in jedem der angegebenen Algorithmen durchgeführte Transport der Strukturen erst am Ende des Sampling-Prozesses durchzuführen, und bis dorthin die Strukturen in einem Zwischenzustand zu belassen. Ein Beispiel für eine derartige Implementierung wird im Rahmen dieser Arbeit noch gegeben.

Bezüglich des Laufzeitverhaltens sehen wir zunächst leicht, dass für iterative Systeme alle angegebenen Sampling-Algorithmen ein Laufzeitverhalten von $O(n)$, wobei n die Strukturgröße bezeichnet, haben. Die Zufallszahlengeneratoren können mittels Inversen-Methode (siehe Abschnitt

4.5) so implementiert werden, dass deren Zeitbedarf $O(k)$ ist, wobei k die erzeugte Zufallszahl bezeichnet. Die Implementierung des Transports der Strukturen kann, wie wir im nächsten Kapitel sehen werden, auch so gestaltet werden, dass das Ummarkieren der Atome in linearem Zeitaufwand funktioniert.

Bei rekursiven Spezifikationen besteht für konstruierbare Spezies auch lineares Laufzeitverhalten, wenn wir die angegebenen Strukturtransporte erst am Ende des Sampling-Prozesses durchführen (dies funktioniert in $O(n)$). In [3, S. 34] wird gezeigt, dass die Größe des Aufrufbaumes $O(mn)$ ist, wobei, m die Anzahl der Gleichungen der Spezifikation und n die Gröse der gesampelten Struktur ist. Schließlich können wir eine Konstante C finden, mit der die Laufzeit pro Knoten dieses Baumes beschränkt ist.

3.3 Größe der erzeugten Strukturen

Im exponentiellen Boltzmann-Modell, das im letzen Abschnitt vorgestellt wurde, besitzt jede Struktur eine positive Wahrscheinlichkeit. Im Artikel [5] wird untersucht, wie Sampler derart gestaltet werden können, dass nur mehr Strukturen mit einer Größe in einem vorgegebenen Intervall (*Approximate-Size-Sampling*) beziehungsweise mit fest vorgegebener Größe (*Exact-Size-Sampling*) erzeugt werden.

Zuerst betrachten wir den Erwartungswert und die Varianz der Größe einer Struktur, die nach dem Boltzmann-Modell zum Parameter x verteilt ist.

Satz 3.3.1. *Sei $\mathcal{F}$ eine analytische Spezies, deren exponentielle erzeugende Funktion $F(z)$ den Konvergenzradius $\rho > 0$ hat, $x \in (0, \rho)$ und $\Gamma\mathcal{F}$ der exponentielle Boltzmann-Sampler von $\mathcal{F}$, dann gilt*

$$\mathbb{E}|\Gamma\mathcal{F}| = x \cdot \frac{F'(x)}{F(x)} \tag{51}$$

$$\operatorname{Var}|\Gamma\mathcal{F}| = x^2 \cdot \frac{F''(x)}{F(x)} + x \cdot \frac{F'(x)}{F(x)} - \Big(x \cdot \frac{F'(x)}{F(x)}\Big)^2 \tag{52}$$

Beweis. Zuerst betrachten wir den Erwartungswert der Größe der vom Sampler erzeugten Struktur. Es gilt für alle $x \in (0, \rho)$

$$F'(x) = \Big(\sum_{n\geq 0} \frac{f_n}{n!} \cdot x^n\Big)' = \sum_{n\geq 1} n \cdot \frac{f_n}{n!} \cdot x^{n-1}, \text{ und}$$

$$F''(x) = \Big(\sum_{n\geq 0} \frac{f_n}{n!} \cdot x^n\Big)'' = \sum_{n\geq 2} n \cdot (n-1) \cdot \frac{f_n}{n!} \cdot x^{n-2}.$$

Daher folgt für den Erwartungswert

$$\mathbb{E}|\Gamma\mathcal{F}| = \sum_{n\geq 0} n \cdot P\big(|\Gamma\mathcal{F}| = n\big) = \sum_{n\geq 1} n \cdot \frac{f_n \cdot x^n}{n! \cdot F(x)} = \frac{x}{F(x)} \sum_{n\geq 1} n \cdot \frac{f_n}{n!} \cdot x^{n-1} = x \cdot \frac{F'(x)}{F(x)}.$$

Für die Varianz gilt dann aufgrund des Steiner'schen Verschiebungssatzes

$$\begin{aligned}
\operatorname{Var}|\Gamma\mathcal{F}| &= \mathbb{E}\big(|\Gamma\mathcal{F}|^2\big) - \big(\mathbb{E}|\Gamma\mathcal{F}|\big)^2 = \sum_{n\geq 0} n^2 \cdot P\big(|\Gamma\mathcal{F}| = n\big) - \big(\mathbb{E}|\Gamma\mathcal{F}|\big)^2 \\
&= \sum_{n\geq 0} \big(n^2 - n\big) P\big(|\Gamma\mathcal{F}| = n\big) + \sum_{n\geq 0} n \cdot P\big(|\Gamma\mathcal{F}| = n\big) - \big(\mathbb{E}|\Gamma\mathcal{F}|\big)^2 \\
&= \sum_{n\geq 2} n\cdot(n-1)\frac{f_n \cdot x^n}{n!\cdot F(x)} + x\cdot\frac{F'(x)}{F(x)} - \Big(x\cdot\frac{F'(x)}{F(x)}\Big)^2 \\
&= x^2 \cdot \sum_{n\geq 2} n\cdot(n-1)\frac{f_n}{n!}\cdot x^{n-2} + x\cdot\frac{F'(x)}{F(x)} - \Big(x\cdot\frac{F'(x)}{F(x)}\Big)^2 \\
&= x^2\cdot\frac{F''(x)}{F(x)} + x\cdot\frac{F'(x)}{F(x)} - \Big(x\cdot\frac{F'(x)}{F(x)}\Big)^2.
\end{aligned}$$

□

Bevor wir uns weiter mit der Größe der gesampelten Strukturen beschäftigen, betrachten wir noch Algorithmus 9, der eine Erweiterung des exponentiellen Boltzmann-Samplers darstellt. Zusätzlich zum Boltzmann-Parameter und der Folge der Grundmenge erwartet der sogenannte *exponentielle Boltzmann-Verwerfungs-Sampler* eine Zielgröße $n \in \mathbb{N}_0$ und eine relative Toleranz $\epsilon \geq 0$, und gibt nur Strukturen s mit $n(1-\epsilon) \leq |s| \leq n(1+\epsilon)$ zurück. Wählen wir $\epsilon > 0$, so handelt es sich um einen *Approximate-Size-Sampler*. Für $\epsilon = 0$ erhalten wir einen sogenannten *Exact-Size-Sampler*, der ausschließlich Strukturen mit Größe n erzeugt.

Algorithmus 9 $\mu\mathcal{F}\big(x, \big(U_n\big)_{n\in\mathbb{N}_0}, n, \epsilon\big)$, exponentieller Boltzmann-Verwerfungs-Sampler der Spezies $\mathcal{F}$

Eingabe: Spezies $\mathcal{F}$ mit exponentiellem Boltzmann-Sampler $\Gamma\mathcal{F}$, Mengenfolge $\big(U_n\big)_{n\in\mathbb{N}_0}$ mit $|U_n| = n$ für alle $n \in \mathbb{N}_0$, Zielgröße $n \in \mathbb{N}_0$ und relative Toleranz $\epsilon \geq 0$.

Ausgabe: Struktur $s \in \bigcup_{n\in\mathbb{N}_0} \mathcal{F}[U_n]$ mit $n(1-\epsilon) \leq |s| \leq n(1+\epsilon)$, die nach dem exponentiellen Boltzmann-Modell bedingt durch die Größeneinschränkung verteilt ist.

repeat
  $s \leftarrow \Gamma\mathcal{F}\big(x, \big(U_n\big)_{n\in\mathbb{N}_0}\big)$
until $n(1-\epsilon) \leq |s| \leq n(1+\epsilon)$
return s.

Definition 3.3.2. Sei $F(x)$ die analytische exponentielle erzeugende Funktion der Spezies F mit Konvergenzradius $\rho > 0$ und

$$v_1(x) := x\cdot\frac{F'(x)}{F(x)},\ v_2(x) := x^2\cdot\frac{F''(x)}{F(x)},$$

dann erfüllt sie die sogenannte *Mittelwertbedingung* genau dann, wenn

$$\lim_{x\to\rho^-} v_1(x) = +\infty. \tag{53}$$

Weiters erfüllt sie die *Varianzbedingung* genau dann, wenn

$$\lim_{x\to\rho^-} \frac{\sqrt{v_2(x) + v_1(x) - v_1^2(x)}}{v_1(x)} = 0. \tag{54}$$

Ist die Mittelwertbedingung erfüllt, so ist es zu jeder Zielgröße n der zu sampelnden Strukturen möglich, über den Boltzmann-Parameter den Erwartungswert der Strukturgröße auf die Zielgröße einzustellen, indem man die Gleichung

$$n = x\frac{F'(x)}{F(x)} \tag{55}$$

nach x auflöst. Diese hat in $(0, \rho)$ genau eine Lösung, die wir mit x_n bezeichnen. Es gilt dann folgender Satz (siehe [5, S. 602]).

Satz 3.3.3. *Sei $\mathcal{F}$ eine analytische Spezies, zu der ein exponentieller Boltzmann-Sampler, der in linearer Zeit sampelt, existiert, und sei $n \in \mathbb{N}_0$ und $\epsilon > 0$. Erfüllt die exponentielle erzeugende Funktion $F(z)$ der Spezies $\mathcal{F}$ die Mittelwert- und die Varianzbedingung, dann konvergiert die Anzahl der benötigten Versuche des Verwerfungs-Sampler $\mu\mathcal{F}\big(x_n, \big(U_n\big)_{n\in\mathbb{N}_0}, n, \epsilon\big)$ in Wahrscheinlichkeit für $n \to \infty$ gegen 1. Insbesonders können Strukturen der Größe n mit der gegebenen Toleranz ϵ in $O(n)$ erzeugt werden.*

Nun betrachten wir eine weitere Klasse von erzeugenden Funktionen, für die der Verwerfungs-Sampler gute Resultate liefert.

Definition 3.3.4. Eine Funktion $f(z)$, die analytisch bei 0 ist und einen endlichen Konvergenzradius $\rho > 0$ besitzt, heißt genau dann Δ-singulär, wenn sie die folgenden zwei Bedingungen erfüllt.

1. Die einzige Singularität am Kreis $\big\{z \in \mathbb{C} : \big|z\big| = \rho\big\}$ ist ρ, und es existieren $r > \rho$ und θ mit $0 < \theta < \frac{\pi}{2}$, sodass f auf der Menge
$$\Delta(r, \theta) := \big\{z \in \mathbb{C} : z \neq \rho, \big|z\big| < r, \arg(z - \rho) \notin (-\theta, \theta)\big\}$$
analytisch fortsetzbar ist.

2. Wenn z innerhalb der Menge Δ gegen ρ strebt, dann existiert eine Darstellung von $f(z)$ der Form
$$f(z) = P(z) + c_0\Big(1 - \frac{z}{\rho}\Big)^{-\alpha} + o\bigg(\Big(1 - \frac{z}{\rho}\Big)^{-\alpha}\bigg) \text{ mit } \alpha \in \mathbb{R} \setminus \{0, -1, -2, \ldots\},$$
wobei $P(z)$ ein Polynom ist. Die Größe $-\alpha$ wird *singulärer Exponent* von $f(z)$ genannt.

Für Spezies mit erzeugenden Funktionen, die diese Bedingungen erfüllen, gilt der folgende in [5, S. 606] bewiesene Satz.

Satz 3.3.5. *Sei $\mathcal{F}$ eine analytische kombinatorische Spezies mit exponentiellen Boltzmann-Sampler, der in linearer Zeit sampelt, und deren exponentielle erzeugende Funktion $F(z)$ Δ-singulär mit einem Exponenten $-\alpha < 0$ ist, dann ist der Erwartungswert der Versuche, die der Verwerfungs-Sampler $\mu\mathcal{F}\big(x_n, (U_n)_{n\in\mathbb{N}_0}, n, \epsilon\big)$ benötigt, um eine Struktur der gewünschten Größe $n \in \mathbb{N}_0$ mit der relativen Toleranz $\epsilon > 0$ zu erzeugen, asymptotisch gleich der Konstanten*
$$\frac{1}{\xi_\alpha(\epsilon)}, \text{ wobei } \xi_\alpha(\epsilon) = \frac{\alpha^\alpha}{\Gamma(\alpha)} \int_{-\epsilon}^{\epsilon} (1 + s)^{\alpha-1} \exp\big(-\alpha(1 + s)\big)\, ds.$$
Insbesonders ist der Zeitbedarf des Approximate-Size-Samplers $O(n)$ und der Zeitbedarf des Exact-Size-Samplers $O(n^2)$.

Die Bedingung der Δ-Singularität wird von vielen in der praxis auftretenden Spezies erfüllt. Allerdings ist der singuläre Exponent $-\alpha$ der exponentiellen erzeugenden Funktion oft größer 0. In diesen Fällen können wir den Punktierungs-Operator (siehe Definition 2.4.13) anwenden. Betrachten wir eine kombinatorische Spezies $\mathcal{F}$, deren exponentielle erzeugende Funktion $F(z)$ Δ-singulär mit Exponenten $-\alpha$ ist, so ist die erzeugende Funktion der punktierten Spezies $\mathcal{F}^\bullet$ ebenfalls Δ-singulär mit Exponenten $-\alpha - 1$. Ausserdem gilt, dass wir durch einfaches Weglassen der durch die Punktierung hinzugefügte Marke einer $\mathcal{F}^\bullet$-Struktur im Prinzip eine $\mathcal{F}$-Struktur erhalten. Die Eigenschaft der Boltzmann-Sampler, dass Strukturen einer Größe gleich wahrscheinlich erzeugt werden, bleibt erhalten.

Bezeichnen wir mit $\mu\mathcal{F}^{\bullet k}\big(x, \big(U_n\big)_{n\in\mathbb{N}_0}, n, \epsilon\big)$ einen Verwerfungs-Boltzmann-Sampler mit Zielgröße $n \in \mathbb{N}_0$ und relativer Toleranz $\epsilon \geq 0$, der Strukturen der k-mal punktierten Spezies $\mathcal{F}$ erzeugt und anschließend die Marken entfernt, so gilt folgender Satz (siehe [5, S. 611]).

Satz 3.3.6. *Sei $\mathcal{F}$ eine analytische kombinatorische Spezies mit in linearer Zeit sampelndem exponentiellem Boltzmann-Sampler, deren exponentielle erzeugende Funktion $F(z)$ Δ-singulär mit einem beliebigen Exponenten $-a \neq \in \{0, 1, \ldots\}$ ist. Sei weiters $\alpha_+ := \max\{0, \lceil -\alpha \rceil\}$ der ganzzahlige positive Teil von $-\alpha$ und $\alpha_0 := \alpha + \alpha_+$ der Nachkommaanteil, dann ist der Ewartungswert der Versuche, die der Sampler $\mu\mathcal{F}^{\bullet\alpha_+}\big(x_n, (U_n)_{n\in\mathbb{N}_0}, n, \epsilon\big)$ benötigt, asymptotisch gleich der Konstanen $\frac{1}{\xi_{\alpha_0}(\epsilon)}$. Insbesonders ist der Zeitbedarf des Approximate-Size-Samplers im Durchschnitt $O(n)$.*

Manche kombinatorische Systeme gestatten die Anwendung einer weiteren Technik. Die exponentielle erzeugende Funktion ist bei diesen Systemen von einer derartigen Gestalt, dass als Boltzmann-Parameter der Konvergenzradius gewählt werden kann. Hier entfällt dann die Ermittlung eines geeigneten Parameters zur gewünschten Strukturgröße. Die Behandlung dieser Systeme und deren sogenannter *singulären Boltzmann-Sampler* würde allerdings den Rahmen dieser Arbeit sprengen und ist in [5, S. 612-620] zu finden.

3.4 Auswertung der erzeugenden Funktion

Im Zuge des Sampling-Prozesses müssen wir die erzeugende Funktion des gegebenen kombinatorischen Systems an der Stelle des Boltzmann-Parameters auswerten. Ein Algorithmus, der das leistet, wird *Boltzmann-Orakel* genannt.

Im Artikel [14] wird eine effiziente Methode dafür vorgestellt, die sowohl zum Auswerten der exponentiellen als auch zum Auswerten der gewöhnlichen erzeugenden Funktion verwendet werden kann. Zuerst wird die aus der Analysis bekannte *Newton-Iteration* in den Kontext der kombinatorischen Spezies übertragen. Dann wird die Konvergenz der Iterationsfolge der Newton-Iteration zuerst auf erzeugende Funktionen, dann auf die numerische Auswertung der erzeugenden Funktionen übertragen. Die gewöhnlichen erzeugenden Funktionen werden im Rahmen dieser Arbeit nicht näher untersucht.

Definition 3.4.1. Sei $\big(\boldsymbol{\mathcal{F}}^{[n]}\big)_{n\geq 0}$ eine konvergente Folge (von Vektoren) von Spezies. Die Konvergenz wird *quadratisch* genannt, wenn sich der Kontakt der Folgenglieder mit dem Grenzwert in jedem Schritt verdoppelt, wenn also gilt, dass aus der Tatsache, dass $\boldsymbol{\mathcal{F}}^{[n]}$ Kontakt der Ordnung k mit $\boldsymbol{\mathcal{F}}$ besitzt, folgt, dass $\boldsymbol{\mathcal{F}}^{[n+1]}$ Kontakt der Ordnung $2k+1$ mit $\boldsymbol{\mathcal{F}}$ hat.

Definition 3.4.2. Sei $\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{H}}\big(\boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{Y}}\big)$ ein wohlfundiertes System. Sein *kombinatorischer Newton-Operator* ist durch

$$\boldsymbol{\mathcal{N}}_{\boldsymbol{\mathcal{H}}}(\boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{Y}}) := \boldsymbol{\mathcal{Y}} + \Big(\sum_{k\geq 0}\Big(\frac{\boldsymbol{\partial\mathcal{H}}}{\boldsymbol{\partial\mathcal{Y}}}(\boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{Y}})\Big)^k\Big) \cdot (\boldsymbol{\mathcal{H}}(\boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{Y}}) - \boldsymbol{\mathcal{Y}}) \tag{56}$$

definiert.

Nun gilt, wie in [14, S. 24-26] bewiesen, der folgende Satz.

Satz 3.4.3. *Sei $\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{H}}\big(\boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{Y}}\big)$ ein wohlfundiertes System, dann ist die Folge, die durch*

$$\boldsymbol{\mathcal{Y}}^{[0]} = \mathbf{0} \text{ und } \boldsymbol{\mathcal{Y}}^{[n+1]} = \boldsymbol{\mathcal{N}}_{\boldsymbol{\mathcal{H}}}\big(\boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{Y}}^{[n]}\big) \text{ für } n \geq 0 \tag{57}$$

gegeben ist, wohldefiniert und konvergiert quadratisch gegen die Lösung $\boldsymbol{\mathcal{S}}$ des Systems mit $\boldsymbol{\mathcal{S}}\big(\mathbf{0}\big) = \boldsymbol{\mathcal{H}}^m\big(\mathbf{0}, \mathbf{0}\big)$.

Bemerkung. Wenn $\boldsymbol{\mathcal{H}}\big(\mathbf{0}, \mathbf{0}\big) \neq 0$, dann kann es sein, dass die ersten Iterationen möglicherweise nicht Kontakt 0 mit $\boldsymbol{\mathcal{S}}$ besitzen. Der Satz besagt aber, dass dies ab einem gewissen Index doch der Fall ist, und dass sich ab diesem Punkt der Kontakt zur Lösung pro Iterationsschritt verdoppelt.

Da die Konvergenzbegriffe von Spezies und formalen Potenzreihen kompatibel sind, kann leicht gefolgert werden, dass folgender Satz [14, S.33] gilt.

Satz 3.4.4. *Sei $\boldsymbol{\mathcal{F}} = \boldsymbol{\mathcal{H}}\big(\mathcal{Z}, \boldsymbol{\mathcal{Y}}\big)$ ein wohlfundiertes System und $\boldsymbol{\mathcal{F}}$ ein Vektor von Spezies mit der exponentiellen erzeugenden Funktion $\mathbf{F}(z)$. Wenn*

$$\boldsymbol{\mathcal{Y}}^{[0]} = \mathbf{0} \text{ und } \boldsymbol{\mathcal{Y}}^{[n+1]} = \boldsymbol{\mathcal{F}}\big(\mathcal{Z}, \boldsymbol{\mathcal{Y}}^{[n]}\big) \text{ für } n \geq 0,$$

eine monoton steigende Folge von Spezies ist, die gegeben die Lösung $\boldsymbol{\mathcal{S}}$ des Systems konvergiert, dann konvergiert die Vektor-Folge

$$\mathbf{Y}^{[0]}(z) := \mathbf{0} \text{ und } \mathbf{Y}^{[n+1]} = \mathbf{F}\big(z, \mathbf{Y}^{[n](z)}\big) \text{ für } n \geq 0,$$

der exponentiellen erzeugenden Funktionen gegen den Vektor der exponentiellen erzeugenden Funktion $\mathbf{S}(z)$ des Spezies-Vektors $\boldsymbol{\mathcal{S}}$.

Definition 3.4.5. Sei $\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{H}}\big(\mathcal{Z}, \boldsymbol{\mathcal{Y}}\big)$ ein wohlfundiertes System. Der *Newton-Operator für exponentielle erzeugende Funktionen* ist durch

$$\mathbf{N}_{\boldsymbol{\mathcal{H}}}\big(z, \mathbf{Y}(z)\big) = \mathbf{Y}(z) + \big(\mathbf{Id} - \boldsymbol{\partial}\mathbf{H}/\boldsymbol{\partial}\mathbf{Y}\big(z, \mathbf{Y}(z)\big)\big)^{-1} \cdot \big(\mathbf{H}\big(z, \mathbf{Y}(z)\big) - \mathbf{Y}(z)\big) \tag{58}$$

definiert, wobei $\mathbf{H}$ den Vektor der exponentiellen erzeugenden Funktionen des Spezies-Vektors $\boldsymbol{\mathcal{H}}$ bezeichnet.

Genauso wie im Fall der Spezies konvergiert auch für exponentielle erzeugende Funktionen die Iterationsfolge des Newton-Verfahrens quadratisch gegen die Lösung des Systems.

Definition 3.4.6. Sei $\big(\mathbf{F}^{[n]}\big)_{n\geq 0}$ eine gegen $\mathbf{F}$ konvergente Folge von (Vektoren von) formalen Potenzreihen. Die Konvergenz wird *quadratisch* genannt, wenn sich der Kontakt der Folgenglieder mit dem Grenzwert in jedem Schritt verdoppelt, wenn also gilt, dass aus der Tatsache, dass $\mathbf{F}^{[n]}$ Kontakt der Ordnung k mit $\mathbf{F}$ besitzt, folgt, dass $\mathbf{F}^{[n+1]}$ Kontakt der Ordnung $2k+1$ mit $\mathbf{F}$ hat.

Satz 3.4.7. *Sei $\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{H}}\big(\mathcal{Z}, \boldsymbol{\mathcal{Y}}\big)$ ein wohlfundiertes System und der Spezies-Vektor $\boldsymbol{\mathcal{S}}$ dessen Lösung. Die* Newton-Iteration*, die durch*

$$\mathbf{Y}^{[0]}(z) = \mathbf{0} \text{ und } \mathbf{Y}^{[n+1]}(z) = \mathbf{N}_{\boldsymbol{\mathcal{H}}}\Big(z, \mathbf{Y}^{[n]}(z)\Big) \text{ für } n \geq 0,$$

definiert ist, konvergiert quadratisch gegen den Vektor $\mathbf{S}(z)$ der exponentiellen erzeugenden Funktionen der Lösung $\boldsymbol{\mathcal{S}}$.

Schließlich kann das Newton-Verfahren auch auf die Auswertung der exponentiellen erzeugenden Funktion der Lösung eines wohlfundierten kombinatorischen Systems transferiert werden. Im Gegensatz zum Verfahren, das aus der Analysis bekannt ist, und bei dem der Startwert in einer geeigneten Umgebung der Lösung liegen muss, kann die Iteration allerdings immer beim Punkt $\mathbf{0}$ beginnen, und konvergiert garantiert gegen die richtige Lösung.

Zuerst wird in [14, S. 42] der folgende Satz bewiesen, der die Konvergenz der durch eine beliebige Spezies definierte Iterationsfolge, die monoton gegen die Lösung eines wohlfundierten Systems konvergiert, auf die entsprechende Iterationsfolge der Auswertungen von exponentiellen erzeugenden Funktionen transferiert.

Satz 3.4.8. *Sei $\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{H}}\big(\mathcal{Z}, \boldsymbol{\mathcal{Y}}\big)$ wohlfundiertes System, und sei weiters $\boldsymbol{\mathcal{F}}$ eine analytische Spezies, sodass die durch*

$$\boldsymbol{\mathcal{Y}}^{[0]} = \mathbf{0} \text{ und } \boldsymbol{\mathcal{Y}}^{[n+1]} = \boldsymbol{\mathcal{F}}\big(\mathcal{Z}, \boldsymbol{\mathcal{Y}}^{[n]}\big) \text{ für } n \geq 0, \tag{59}$$

definierte Iterationsfolge eine monoton steigende Folge von Spezies ist, die gegen die Lösung $\boldsymbol{\mathcal{S}}$ des Systems konvergiert, dann folgt, dass die exponentielle erzeugende Funktion $\mathbf{S}(z)$ von $\boldsymbol{\mathcal{S}}$ einen positiven Konvergenzradius ρ besitzt, und dass für alle α mit $|\alpha| < \rho$ die Folge

$$\mathbf{y}^{[0]} = \mathbf{0} \text{ und } \mathbf{y}^{[n+1]} = \mathbf{F}\big(\alpha, \mathbf{y}^{[n]}\big) \text{ für } n \geq 0, \tag{60}$$

gegen $\mathbf{S}(\alpha)$ konvergiert.

Da der Newton-Operator für analytisches $\boldsymbol{\mathcal{H}}$ analytisch ist, und die Iterationsfolge laut Satz 3.4.3 die Voraussetzungen von Satz 3.4.8 erfüllt, gilt also auch die folgende Aussage.

Satz 3.4.9. *Sei $\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{H}}\big(\mathcal{Z}, \boldsymbol{\mathcal{Y}}\big)$ ein wohlfundiertes System, wobei $\boldsymbol{\mathcal{H}}$ ein Vektor von analytischen Spezies ist. Sei weiters $\alpha > 0$ innerhalb des Konvergenzradius der exponentiellen erzeugenden Funktion $\mathbf{S}(z)$ der Lösungsspezies $\boldsymbol{\mathcal{S}}$, dann konvergiert die durch*

$$\mathbf{y}^{[0]} = \mathbf{0} \text{ und } \mathbf{y}^{[n+1]} = \mathbf{y}^{[n]} + \Big(\mathbf{Id} - \frac{\boldsymbol{\partial}\mathbf{H}}{\boldsymbol{\partial}\mathbf{Y}}\big(\alpha, \mathbf{y}^{[n]}\big)\Big)^{-1} \cdot \big(\mathbf{H}\big(\alpha, \mathbf{y}^{[n]}\big) - \mathbf{y}^{[n]}\big), \tag{61}$$

definierte Iterationsfolge gegen $\mathbf{S}(\alpha)$.

Dieses Verfahren wird in der im nächsten Kapitel vorgestellten Implementierung als Boltzmann-Orakel verwendet.

4 Implementierung

In diesem Kapitel werden wir die Implementierung eines exponentiellen Boltzmann-Samplers in der Programmiersprache *Java* betrachten. Der Funktionsumfang des Programms umfasst folgende Punkte:

- Es können in einer XML-Textdatei kombinatorische Spezifikation definiert werden. Diese Spezifikationen bestehen aus beliebig vielen Gleichungen unter Verwendung der Summen-, Produkt-, Folgen-, Mengen- und Zyklen-Spezies, sowie der Spezies der Charakteristik der leeren Menge und der Singleton-Spezies. Weiters können Strukturen mit frei definierbaren Marken versehen werden, die dann bei der Ausgabe der Struktur entsprechend berücksichtigt werden.
- Beim Programmstart kann die zu sampelnde Spezifikation und der Boltzmann-Parameter angegeben werden. Weiters ist es möglich, den Bereich, in dem die Größe der Strukturen liegt, einzuschränken, sowie einzustellen, wie viele Strukturen mit der Zielgröße erzeugt werden sollen.
- Die Ausgabe der Strukturen erfolgt entweder in eine Textdatei oder in die Standardausgabe des Programms.
- Das Programm ist derart strukturiert, dass es möglichst einfach erweiterbar ist, sowie in möglichst vielen Anwendungsbereichen eingesetzt werden kann.

Wie diese Punkte realisiert wurden, werden wir in den folgenden Unterabschnitten sehen. Zuerst werden wir betrachten, wie der Programmaufruf funktioniert und welche Argumente angegeben werden können. Anschliessend betrachten wir das Modul, das für die kombinatorischen Spezifikationen zuständig ist. Dort wird auch beschrieben, wie das Dateiformat für die Spezifikationen aussieht.

Für die Implementierung des Boltzmann-Orakels wird die Newton-Iteration, die in Abschnitt 3.4 theoretisch behandelt wurde, verwendet. Dafür wird ein Löser für lineare Gleichungssysteme benötigt. Wir beschäftigen uns im Anschluss an die Spezifikation daher mit dem Package für lineare Algebra, das im dafür notwendigen Umfang implementiert wurde, und das wir dann auch gleich für das Boltzmann-Orakel, dem der darauf folgende Abschnitt 4.4 gewidmet ist, verwenden.

Um die Strukturen zu sampeln, benötigen wir Generatoren, die Zufallszahlen nach bestimmten Verteilungen erzeugen. Diese Generatoren werden in Abschnitt 4.5 betrachtet. Schließlich folgt in Abschnitt 4.6 die Beschreibung der Implementierung des eigentlichen Boltzmann-Samplers. Abschließend wird im letzten Abschnitt dieses Kapitels noch das Ausgabemodul vorgestellt, das Strukturen in einem Textformat ausgeben kann.

Der Quellcode ist in der Online-Version dieser Arbeit im Anhang vollständig abgedruckt zu finden. In diesem Kapitel werden wir uns mit UML-Klassen-Diagrammen, die die Interfaces, Klassen und deren gegenseitigen Beziehungen sowie alle veröffentlichten Methoden mit Kurzbeschreibung darstellen, begnügen. Eine genauere Dokumentation der Klassen ist als Javadoc-Kommentar im Quellcode zu finden.

4.1 Aufruf des Programms

Der Haupteinstiegspunkt ins Programm ist die Methode *main* der Klasse *ConsoleMain* des Packages *at.techmath.boltzmann.app.console*. Beim Programmaufruf können hierbei diverse Argumente angegeben werden, die steuern, welche Strukturen der Boltzmann-Sampler erzeugt, und in welcher Art diese ausgegeben werden. Die Argumente werden in Tabelle 2 aufgelistet. Die Klasse *CommandLineConfiguration* ist dafür zuständig, die Kommandozeilenargumente zu parsen und zur weiteren Verwendung zur Verfügung zu stellen, beziehungsweise entsprechende Fehlermeldungen zu generieren, falls die Argumente fehlerhaft sind. Je nach Konfiguration werden dann in der *ConsoleMain*-Klasse die Module aus den anderen Packages, die in den folgenden Abschnitten beschrieben sind, parametriert und aufgerufen.

Switch	**Optional**	**Beschreibung**
`-d <Pfad>`	nein	Pfad der Datei, die die Spezifikation beinhaltet.
`-n <Name>`	ja	Name der kombinatorischen Spezifikation. Falls dieses Argument nicht angegeben wird, dann wird die in der Spezifikationsdatei hinterlegte Standard-Spezifikation verwendet. Falls diese ebenfalls nicht angegeben ist, wird eine Fehlermeldung zurückgegeben.
`-s <Name>`	ja	Name der Spezies der Spezifikation. Falls dieses Argument nicht angegeben wird, dann wird die in der verwendeten Spezifikation hinterlegte Standard-Spezies verwendet. Falls diese ebenfalls nicht angegeben ist, wird eine Fehlermeldung zurückgegeben.
`-x <Zahl>`	nein	Boltzmann-Parameter, mit dem gesampelt werden soll. Die Zahl, die angegeben wird, ist eine Gleitkommazahl.
`-m <Modus>`	ja	Ausgabemodus. Es stehen die folgenden Modi zur Wahl: • `text ...` Ausgabe der Strukturen im Textformat, das im Abschnitt über die Ausgabe beschrieben wird. Wenn der Modus nicht angegeben wird, wird dieser standardmäßig dieser Modus verwendet. • `totalsize ...` Ausgabe der Größe der Strukturen. • `oracle ...` Es wird gar nicht gesampelt, sondern lediglich das Boltzmann-Orakel für den angegebenen Boltzmann-Parameter aufgerufen und der Wert der exponentiellen erzeugenden Funktion ausgegeben.
`-f <Pfad>`	ja	Pfad der Datei, in die die Ausgabe erfolgen soll. Falls die Datei schon existiert, wird die Ausgabe am Ende der Datei angehängt. Falls keine Datei angegeben wird, dann erfolgt die Ausgabe in die Standardausgabe des Systems.

Tabelle 2: Kommandozeilenargumente Spezifikation, Boltzmann-Parameter, Ausgabemodus

Switch	Optional	Beschreibung
`-min <Zahl>`	ja	Mindestgröße der gesampelten Strukturen. Falls dieser Parameter nicht angegeben wird, wird als Standardwert 0 verwendet.
`-max <Zahl>`	ja	Maximalgröße der gesampelten Strukturen. Falls dieser Parameter nicht angegeben wird, gibt es keine Beschränkung der Größe nach oben.
`-c <Zahl>`	ja	Anzahl der zu sampelnden Strukturen. Der Standardwert ist 1.
`-t <Zahl>`	ja	Maximalanzahl der Versuche, die gewünschte Anzahl an Strukturen zu sampeln. Wenn die Versuchsanzahl überschritten wird, bevor die angegebene Anzahl an zu sampelnden Strukturen generiert wurde, wird der Sampling-Vorgang abgebrochen. Falls dieser Parameter nicht angegeben wird, gibt es es keine Beschränkung der Versuchsanzahl.

Tabelle 3: Kommandozeilenargumente Strukturgröße und -anzahl

4.2 Kombinatorische Spezifikation

In diesem Abschnitt werden die Interfaces und Klassen beschrieben, die für die Repräsentation und das Laden von kombinatorischen Spezifikationen aus entsprechenden Dateien zuständig sind. Diese finden wir im Package *at.techmath.boltzmann.specification*. Zuerst werden wir die Interfaces betrachten, die definieren, wie eine Spezifikation auszusehen hat, anschließend wird die in dieser Implementierung verwendete konkrete Realisierung dieser Interfaces, die kombinatorische Spezifikationen aus XML-Dateien laden kann, vorgestellt.

4.2.1 Modellierung kombinatorischer Spezifikationen

Die Interfaces, die die kombinatorischen Spezifikationen, so wie sie in den anderen Modulen verwendet werden, modellieren, sind in Abbildungen 9, 10 und 11 dargestellt. Dadurch, dass die anderen Module nur auf diese Interfaces zugreifen, nicht aber auf eine konkrete Implementierung, ist es leicht möglich, das Programm dahingehend zu erweitern, dass die zu sampelnde Spezifikation aus einer anderen Quelle kommt. Sie könnte etwa aus der Grammatik einer kontextfreien Sprache erzeugt werden. In diesem Fall muss nur eine weitere Realisierung der Interfaces implementiert werden, aber die restlichen Module nicht adaptiert werden.

Wie wir in Abbilding 9 erkennen können, werden die kombinatorische Spezifikationen, die durch das Interface *ISpecification* repräsentiert werden, in einer Spezifikationssammlung zusammengefasst. Diese wird durch das Interface *ISpecificationCollection* repräsentiert. Jede Spezifikation wird durch einen eindeutigen Namen identifiziert. Durch diesen Namen kann auf die Spezifikation durch die Spezifikationssammlung zugegriffen werden. Weiters besteht die Möglichkeit, eine sogenannte Standard-Spezifikation innerhalb der Sammlung hervorzuheben, die in dieser konkreten Implementierung beispielsweise verwendet wird, falls das Programm ohne Angabe einer konkreten Spezifikation aufgerufen wird.

Eine kombinatorische Spezifikation besteht aus Spezies (durch das *ISpezies*-Interfaces) repräsentiert. Diese Spezies unterteilen sich in erster Stufe in die Spezies der Charakteristik der leeren Menge (*ISpeciesOne*), die Singleton-Spezies (*ISpeciesSingleton*) und sogenannte *benannte Spezies* (*INamedSpecies*), die jeweils eine Gleichung des kombinatorischen Systems repräsentieren. Die ersten beiden Spezies sind in der kombinatorischen Spezifikation implizit immer genau einmal vorhanden, von den benannten Spezies können beliebig viele existieren.

In Abbildung 10 sehen wir die Arten der Spezies, die in der kombinatorischen Spezifikation vorkommen können. Unter die benannten Spezies fallen die Summen von Spezies (*ISpeciesSum*), das Produkt von Spezies (*ISpeciesProduct*) sowie die Basisspezies der endlichen Folgen (*ISpeciesSequence*), der Mengen (*ISpeciesSet*) und der Zyklen (*ISpeciesCycle*). In diese Spezies werden andere Spezies der Spezifikation eingesetzt. Dies wird durch sogenannte Spezies-Referenzen (*ISpeciesReference*) modelliert. Eine Spezies-Referenz beinhaltet einerseits eine Referenz auf die Spezies, die eingesetzt werden soll, andererseits eine Referenz auf ein Label, mit die Strukturen der eingesetzten Spezies versehen werden sollen. In die Summen- und Produktspezies können mehrere Spezies eingesetzt werden, in die Folgen-, Mengen- und Zyklen-Spezies nur eine. Die Spezies-Referenz kann dabei sowohl auf benannte Spezies, als auch auf die implizit vorhandene Spezies der Charakteristik der leeren Menge beziehungsweise auf die Singleton-Spezies verweisen. Schließlich kann bei der Spezifikation noch eine Standard-Spezies angegeben werden, die in der konkret vorliegenden Implementierung gesampelt wird, falls keine zu sampelnde Spezies angegeben wurde.

Schließlich wird in Abbildung 11 dargestellt, wie das Labeling von Strukturen modelliert ist. Es besteht prinzipiell die Möglichkeit, gesampelte Strukturen mit einer Marke zu versehen. Diese Marken werden durch das Interface *ILabel* repräsentiert und sind in einer Label-Sammlung zusammengefasst (*ILabelCollection*). Jede Spezifikation beinhaltet genau eine Label-Sammlung. Es gibt zwei Arten von Marken. Die erste Art sind ganzzahlige Marken (*IIntegerLabel*), die zwei-

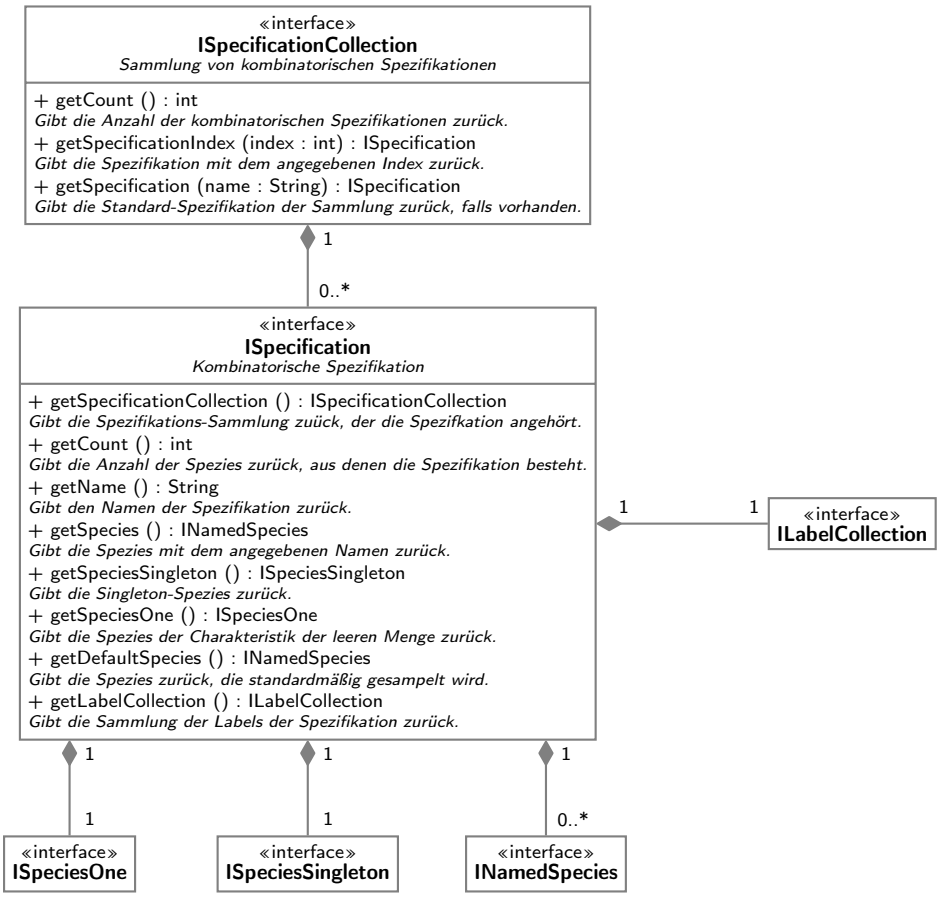

Abbildung 9: Kombinatorische Spezifikation

te Art sind Textmarken (*IStringLabel*). Je nach Anwendung können diese Marken dann in der Ausgabe der Strukturen sinnvoll verwendet werden. Beispielsweise kann man beim Sampeln von Sprachen die Atome mit Marken versehen, die die einzelenen Elemente des Alphabets der Sprache repräsentieren.

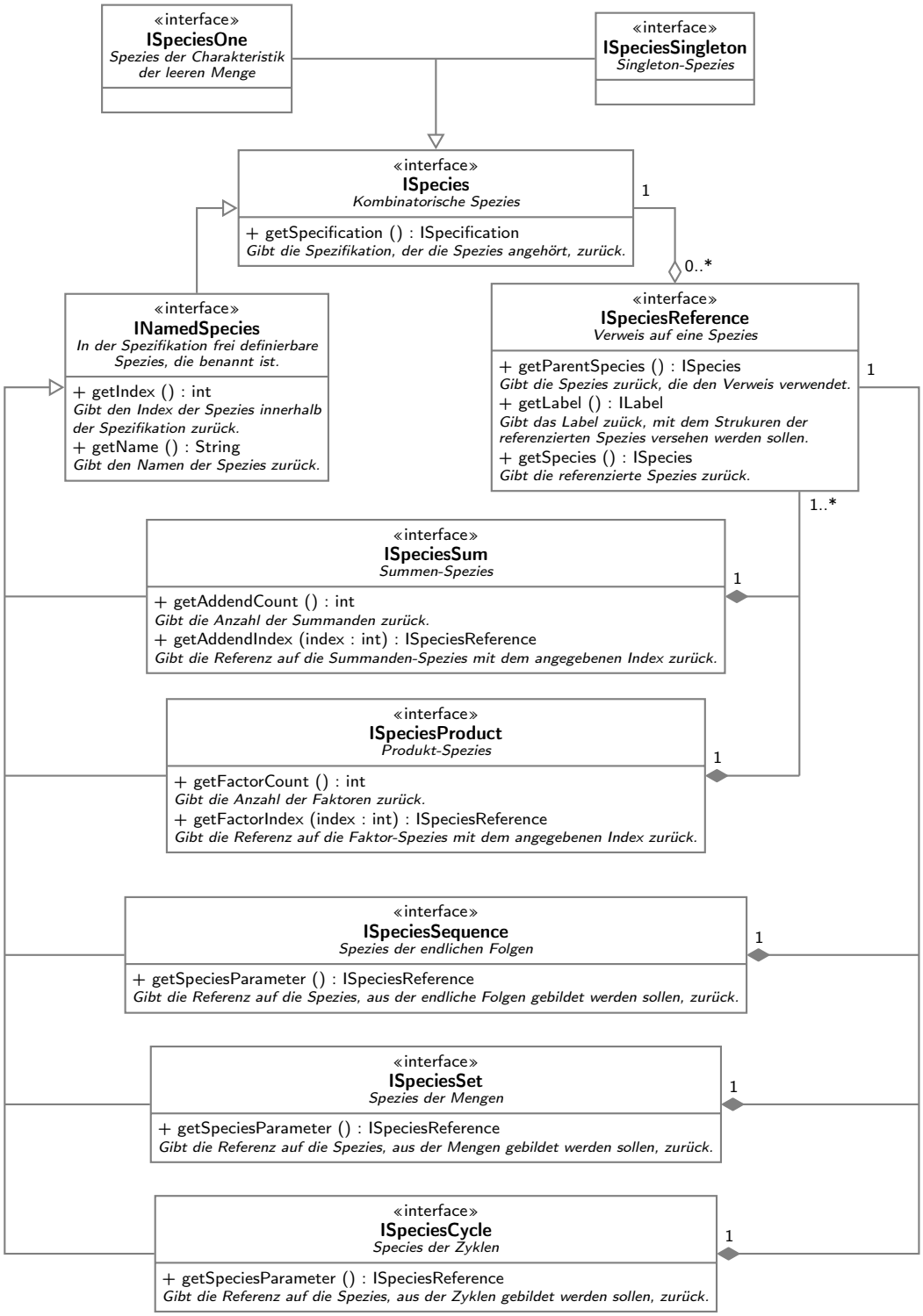

Abbildung 10: Species-Interfaces

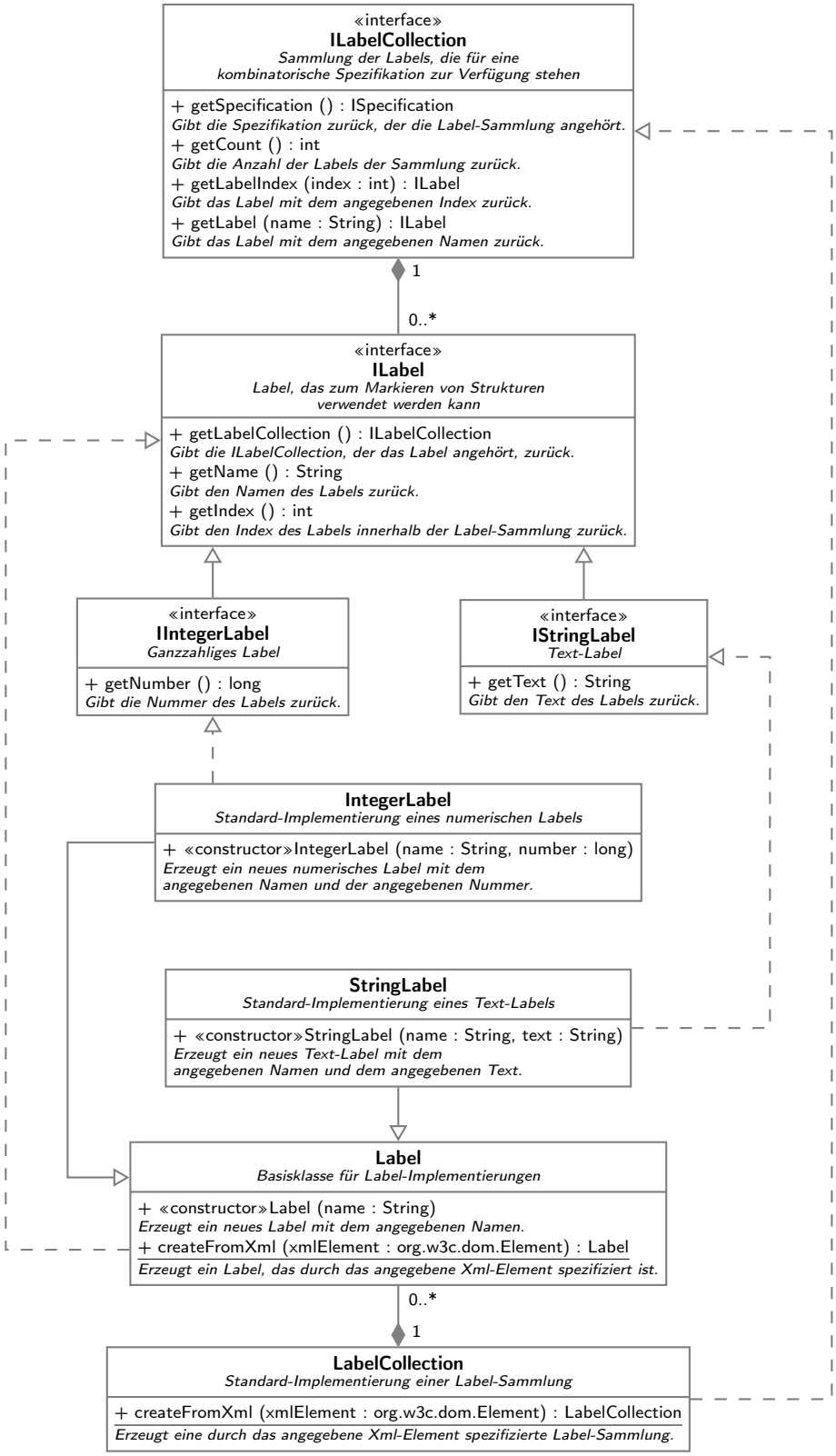

Abbildung 11: Labels für die Strukturen

4.2.2 Laden einer Spezifikationssammlung aus einer XML-Datei

In der durchgeführten Implementierung wurde die Interface-Struktur derart realisiert, dass es möglich ist, eine Spezifikationssammlung aus einer XML-Datei zu laden. Der XML-Standard ist ein Textdatei-Format, das es ermöglicht, hierarchische Strukturen zu speichern, und das sehr gut mit objektorientierten Konzepten vereinbar ist. Weiters gibt es in allen gängigen Programmiersprachen Libraries zum Parsen von XML-Dateien. Der Standard wurde vom World Wide Web-Consortium entwickelt, die Spezifikation ist unter `http://www.w3.org/XML/` zu finden.

Wie wir in den Abbildungen 12 und 13 sehen, ist die Klassenstruktur genauso beschaffen wie die bereits vorgestellte Interface-Struktur. Jede Klasse verfügt über eine statische Methode, die eine Instanz aus einem übergebenen XML-Element erzeugen kann. Jeder Klasse entspricht hier ein eindeutiger XML-Elementname. Die einzelnen Klassen beziehungsweise XML-Elemente werden im weiteren Verlauf noch Vorgestellt.

Als XML-Parser wird hierbei der von der Java-Klassenbibliothek zur Verfügung gestellte Parser verwendet. Die einzige Klasse, die außerhalb des Packages sichtbar ist, ist die *SpecificationCollectionXmlLoader*-Klasse. Diese stellt Methoden zur Verfügung, die es erlauben, die Spezifikation im XML-Format entweder aus einer Datei oder aus einem Stream zu laden.

Zur Implementierung der Spezies-Referenzen, die in Abbildung 14 dargestellt ist, muss noch angemerkt werden, dass drei Typen von Referenzen, nämlich Referenzen auf die Spezies der Charakteristik der leeren Menge (*SpeciesOneReference*), Referenzen auf die Singleton-Spezies (*SpeciesSingletonReference*) und Referenzen auf benannte Spezies (*NamedSpeciesReference*). Dies ist notwendig, da die ersten beiden Spezies innerhalb der Spezifikation keinen Namen besitzen und diese Art der Implementierung eine Möglichkeit darstellt, sie trotzdem zu referenzieren.

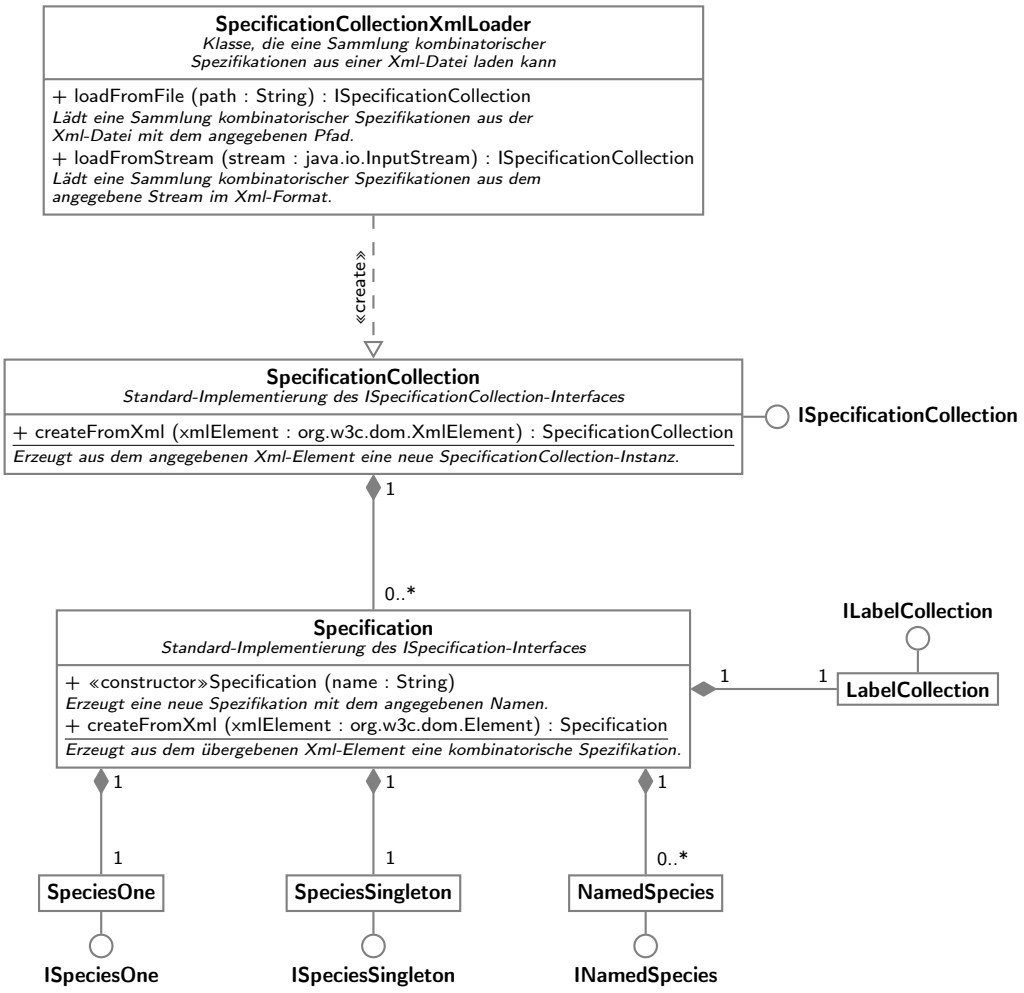

Abbildung 12: Implementierung der kombinatorischen Spezifikationen

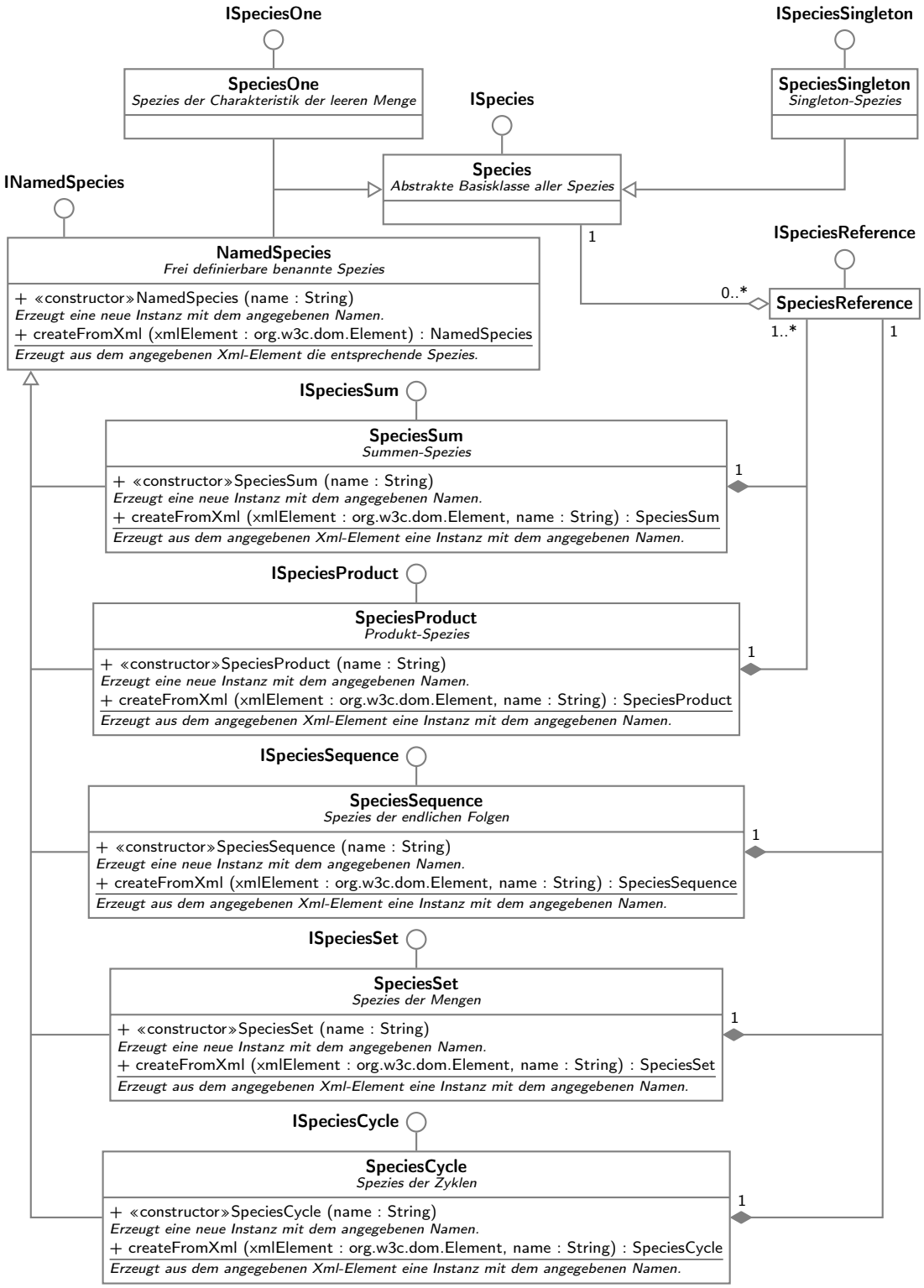

Abbildung 13: Spezies-Implementierung

Abbildung 14: Spezies-Referenzen

Nun folgt eine Auflistung aller XML-Elemente und der zugehörigen Klassen mit der Angabe aller Attribute und erlaubten Unterelemente. Jedes Element befindet sich im XML-Namespace `http://techmath.at/boltzmann/specification/2014/11/06`.

Element `specifications` - Klasse *SpecificationCollection*

Hierbei handelt es sich um eine Sammlung von kombinatorischen Spezifikationen.

Unterlemente

Element	Anzahl	Beschreibung
specification	≥ 0	Spezifikationen, die in der Sammlung enthalten sind.

Attribute

Name	Typ	Optional	Beschreibung
defaultspecification	String	ja	Name der Standard-Spezifikation. Wenn nicht angegeben, dann gibt es keine Standard-Spezifikation.

Element `specification` - Klasse *Specification*

Kombinatorische Spezifikation. Die Spezifikation besteht aus den in den Unterelementen angegebenen Spezies. Hierbei muss mindestens eine Spezies angegeben werden.

Unterlemente

Element	Anzahl	Beschreibung
sum	≥ 0	Summen-Spezies
product	≥ 0	Produkt-Spezies
sequence	≥ 0	Spezies endlicher Folgen
set	≥ 0	Mengen-Spezies
cycle	≥ 0	Zyklen-Spezies
labels	$0 \ldots 1$	Sammlung der Labels

Attribute

Name	Typ	Optional	Beschreibung
name	String	nein	Name der Spezifikation. Muss innerhalb der Spezifikations-Sammlung eindeutig sein.
defaultspecies	String	ja	Name der Standard-Spezies. Wenn nicht angegeben, dann gibt es keine Standard-Spezies.

Element `labels` - Klasse *LabelCollection*

Sammlung von Labels.

Unterlemente

Element	Anzahl	Beschreibung
integerlabel	≥ 0	Ganzzahliges Label
textlabel	≥ 0	Text-Label

Element `integerlabel` - Klasse *IntegerLabel*

Ganzzahliges Label. Die Zahl muss als Element-Inhalt angegeben werden.

Attribute

Name	Typ	Optional	Beschreibung
name	String	nein	Name des Labels. Muss innerhalb der Label-Sammlung eindeutig sein.

Element `textlabel` - Klasse *StringLabel*

Text-Label. Der Text des Labels muss als Element-Inhalt angegeben werden.

Attribute

Name	Typ	Optional	Beschreibung
name	String	nein	Name des Labels. Muss innerhalb der Label-Sammlung eindeutig sein.

Element `sum` - Klasse *SpeciesSum*

Summen-Spezies, in welche die durch die untergeordneten Speziesreferenz-Elemente definierte Spezies eingesetzt werden.

Unterlemente

Element	Anzahl	Beschreibung
one	≥ 0	Referenz auf die Spezies der Charakteristik der leeren Menge.
atom	≥ 0	Referenz auf die Singleton-Spezies.
ref	≥ 0	Referenz auf eine benannte Spezies der Spezifikation.

Attribute

Name	Typ	Optional	Beschreibung
name	String	nein	Name der Spezies. Muss innerhalb der Spezifikation eindeutig sein.

Element `product` - Klasse *SpeciesProduct*

Produkt-Spezies, in welche die durch die untergeordneten Speziesreferenz-Elemente definierte Spezies eingesetzt werden.

Unterlemente

Element	Anzahl	Beschreibung
one	≥ 0	Referenz auf die Spezies der Charakteristik der leeren Menge.
atom	≥ 0	Referenz auf die Singleton-Spezies.
ref	≥ 0	Referenz auf eine benannte Spezies der Spezifikation.

Attribute

Name	Typ	Optional	Beschreibung
name	String	nein	Name der Spezies. Muss innerhalb der Spezifikation eindeutig sein.

Element `sequence` - Klasse *SpeciesSequence*

Spezies der endlichen Folgen, in welche die durch das untergeordnete Speziesreferenz-Element definierte Spezies eingesetzt wird. Es muss genau ein Referenz-Element angegeben werden.

Unterlemente

Element	Anzahl	Beschreibung
atom	0..1	Referenz auf die Singleton-Spezies.
ref	0..1	Referenz auf eine benannte Spezies der Spezifikation.

Element `set` - Klasse *SpeciesSet*

Mengen-Spezies, in welche die durch das untergeordnete Speziesreferenz-Element definierte Spezies eingesetzt wird. Es muss genau ein Referenz-Element angegeben werden.

Unterlemente

Element	Anzahl	Beschreibung
atom	0..1	Referenz auf die Singleton-Spezies.
ref	0..1	Referenz auf eine benannte Spezies der Spezifikation.

Attribute

Name	Typ	Optional	Beschreibung
name	String	nein	Name der Spezies. Muss innerhalb der Spezifikation eindeutig sein.

Element `cycle` - Klasse *SpeciesCycle*

Zyklen-Spezies, in welche die durch das untergeordnete Speziesreferenz-Element definierte Spezies eingesetzt wird. Es muss genau ein Referenz-Element angegeben werden.

Unterlemente

Element	Anzahl	Beschreibung
atom	0..1	Referenz auf die Singleton-Spezies.
ref	0..1	Referenz auf eine benannte Spezies der Spezifikation.

Attribute

Name	Typ	Optional	Beschreibung
name	String	nein	Name der Spezies. Muss innerhab der Spezifikation eindeutig sein.

Element `one` - Klasse *SpeciesOneReference*

Referenz auf die Spezies der Charakteristik der leeren Menge. Optional können referenzierte Strukturen mit einem Label aus der Labelsammlung der Spezifikation versehen werden.

Attribute

Name	Typ	Optional	Beschreibung
label	String	ja	Name des Labels aus der Labelsammlung, mit dem Strukturen der referenzierten Spezies versehen werden sollen. Wenn dieses Attribut nicht angegeben wird, dann erfolgt keine Markierung der Strukturen.

Element `atom` - Klasse *SpeciesSingletonReference*

Referenz auf die Singleton-Spezies. Optional können referenzierte Strukturen mit einem Label aus der Labelsammlung der Spezifikation versehen werden.

Attribute

Name	Typ	Optional	Beschreibung
label	String	ja	Name des Labels aus der Labelsammlung, mit dem Strukturen der referenzierten Spezies versehen werden sollen. Wenn dieses Attribut nicht angegeben wird, dann erfolgt keine Markierung der Strukturen.

Element `ref` - Klasse *NamedSpeciesReference*

Referenz auf eine benannte Spezies der Spezifikation. Optional können referenzierte Strukturen mit einem Label aus der Labelsammlung der Spezifikation versehen werden.

Attribute

Name	Typ	Optional	Beschreibung
name	String	nein	Name der referenzierten Spezies der Spezifikation.
label	String	ja	Name des Labels aus der Labelsammlung, mit dem Strukturen der referenzierten Spezies versehen werden solle. Wenn dieses Attribut nich angegeben wird, dann erfolgt keine Markierung der Strukturen.

Beispiel 4.2.1. Nun sehen wir uns noch ein Beispiel einer Spezifikationssammlung an. Die im folgenden Listing dargestelle XML-Spezifikationssammlung enthält zwei Spezifikationen. Einerseits Binärbäume, bei denen auf die externen Knoten kein Atom verteilt wird, und andererseits Cayley-Bäume. Wir sehen auch, dass die Cayley-Bäume als Standard-Spezifikation gesetzt wurden. Daher wird beim Programmaufruf, wenn nicht explizit eine andere Spezifikation angegeben wurde, ein Cayley-Baum gesampelt.

Die Binärbäume entsprechen der kombinatorischen Spezifikation

$$\mathcal{B} = \mathbf{1} + \mathcal{B} \cdot \mathcal{Z} \cdot \mathcal{B}.$$

Da in der Spezifiaktionssprache allerdings in dieser Version der Implementierung keine komplizierteren Terme angegeben werden können, müssen wir das System auf zwei Gleichungen aufteilen. Dabei wird $\mathcal{B}_1$ als Standard-Spezies der Spezifikation angegeben, und wird daher ohne Angabe einer alternativen Spezies standardmäßig gesampelt.

$$\mathcal{B}_1 = \mathbf{1} + \mathcal{B}_2$$
$$\mathcal{B}_2 = \mathcal{B}_1 \cdot \mathcal{Z} \cdot \mathcal{B}_1$$

Die erwähnte Markierung der internen und externen Knoten geschieht dadurch, dass im Listing in der Labelsammlung zwei Textlabel definiert wurden, welche dann jeweils bei den Referenzen auf die Spezies der Charakteristik der leeren Menge beziehungsweise der Singleton-Spezies angegeben wurden.

Die zweite Spezifikation, die wir im Listing sehen, sind sogenannte Cayley-Bäume. Diese besitzen die kombinatorische Spezifikation

$$\mathcal{G} = \mathcal{Z} \cdot \textsc{Set}(\mathcal{G}).$$

Auch hier müssen wir das System wieder etwas umschreiben, um es in der verwendeten Spezifikationssprache darstellen zu können. Die Spezies $\mathcal{G}_1$ ist hier wiederum als Standard-Spezies angegeben.

$$\mathcal{G}_1 = \mathcal{Z} \cdot \mathcal{G}_2$$
$$\mathcal{G}_2 = \textsc{Set}(\mathcal{G}_1)$$

Bei dieser Spezifikation wird keine zusätzliche Markierung von Strukturen durchgeführt. Daher fallen sowohl die Labelsammlung als auch die Referenzen auf Labels weg.

Listing 1: Beispielspezifikation

```
<?xml version="1.0"?>
<specifications xmlns="http://techmath.at/boltzmann/specification/2014/11/06"
  defaultspecification="CayleyTree">

  <!-- Binaerbaeume mit externen Knoten, auf die kein Atom verteilt wird -->
  <specification name="BinaryTree" defaultspecies="b1">
    <sum name="b1">
      <one label="extern"/>
      <ref name="b2"/>
    </sum>
    <product name="b2">
      <ref name="b1"/>
      <atom label="intern"/>
      <ref name="b1"/>
    </product>
    <labels>
      <textlabel name="extern">Externer Knoten</textlabel>
      <textlabel name="intern">Interner Knoten</textlabel>
    </labels>
  </specification>

  <!-- Cayley-Baeume -->
  <specification name="CayleyTree" defaultspecies="g1">
    <product name="g1">
      <atom/>
      <ref name="g2"/>
    </product>
    <set name="g2">
      <ref name="g"/>
    </set>
  </specification>
</specifications>
```

4.3 Lösen linearer Gleichungssysteme

Für die Implementierung der Newton-Iteration, die für das Boltzmann-Orakel verwendet wird, müssen lineare Gleichungssysteme gelöst werden. Die dafür notwendigen Interfaces und Klassen finden wir im Package *at.techmath.boltzmann.linearalgebra*.

In Abbildung 15 sehen wir Interfaces, die diverse Ausprägungen von Vektoren repräesentieren. Das allgemeinste Interfaces *IRealVector* stellt einen Vektor dar, der aus reellen Komponenten besteht. Es wird einerseits spezialisiert zu Zeilen- beziehungsweise Spaltenvektoren (*IRealRowVector* und *IRealColumnVector*), andererseits zu Vektoren, deren Komponenten verändert werden können (*IMutableRealVector*). Schließlich landen wir bei Interfaces, die veränderbare Zeilen- beziehungsweise Spaltenvektoren repräsentieren (*IMutableRealRowVector* und *IMutableRealColumnVector*).

Die entsprechenden Implementierungen der Vektor-Interfaces sind in Abbildung 16 dargestellt. Es gibt eine Zeilenvektor-Klasse *RealRowVector*, die das Interface *IMutableRealRowVector* realisiert, und eine Spaltenvektor-Klasse, die das Interface *IMutableRealColumnVector* realisiert. Beide Implementierungen sind im Prinzip gleich. Die Einträge der Vektoren werden einfach in einem Array von Gleitkommazahlen mit doppelter Präzision gespeichert, dessen Dimension der Dimension der Vektoren entspricht.

Matrizen sind weitere Objekte aus der linearen Algebra, die wir benötigen. Die entsprechenden Interfaces sind in Abbildung 17 dargestellt. Das Interface *IRealMatrix* stellt eine allgemeine Matrix mit reellen Einträgen dar. Es gibt drei Unterinterfaces. Das erste Unterinterface *IMutableRealMatrix* stellt eine Matrix dar, in der jeder Eintrag individuell verändert werden kann. Das Interface *IRealDiagonalMatrix* repräsentiert eine Diagonalmatrix, also eine quadratische Matrix (a_{ij}), sodass $a_{ij} = 0$ für alle $i \neq j$. Die dritte Art von Matrizen sind Permutationsmatrizen (*IRealPermutationMatrix*), hierbei handelt es sich um quadratische Matrizen. Jeder Permutationsmatrix der Dimension n ist in umkehrbar eindeutiger Weise eine Permutation $\sigma \in S_n$ zugeordnet. Die Matrix hat die Gestalt $\left(\delta_{i\sigma(j)}\right)$.

Die entsprechenden Implementierungen der Matrix-Interfaces sehen wir in Abbildung 17. Die Klasse *RealMatrix* implementiert das *IMutableRealMatrix*-Interface. Die Einträge werden hier in einem Array der Dimension $n \cdot m$ gespeichert, wobei n die Zeilenanzahl und m die Spaltenanzahl bezeichnet. Die Klasse, die das *IRealDiagonalMatrix*-Interface implementiert, heißt *RealDiagonalMatrix*. Hier wird nur die Diagonale in einem Array gespeichert, dessen Dimension der Dimension der Matrix entspricht. Schließlich implementiert die Klasse *RealPermutationMatrix* das Interface *IRealPermutationMatrix*. Hier wird die zur Matrix gehörende Permutation in einem Array in einzeiliger Darstellung gespeichert. Die einzelnen Matrix-Operation sind so programmiert, dass aufgrund der Kombination von Matrix-Typen, die vorkommen, der jeweils effizienteste Algorithmus verwendet wird. Beispielsweise wird bei der Multiplikation einer allgemeinen Matrix mit einer Permutationsmatrix einfach nur eine entsprechende Vertauschung der Zeilen bzw. Spalten durchgeführt, bei der Multiplikation mit einer Diagonalmatrix jeder Spalten- bzw. Zeilen-Vektor einfach mit dem entsprechenden Skalar multipliziert.

Auf diesen Vektor- und Matrix-Typen aufbauend existieren zur Lösung von linearen Gleichungssystemen Interfaces und Klassen, die eine QR-Zerlegung definieren beziehungsweise implementieren. Es wird eine QR-Zerlegung mit Pivotisierung verwendet, wie sie in jedem guten Buch über Numerik zu finden ist. Die orthogonale Matrix wird hierbei mittels Householder-Transformationen erzeugt. Das Ergebnis der QR-Zerlegung einer Matrix A ist eine Darstellung der Form

$$A \cdot P = Q \cdot R,$$

wobei P eine Permutationsmatrix, Q eine orthogonale Matrix und R eine obere Dreiecksmatrix bezeichnet. Die Permutationsmatrix ist deswegen notwendig, da eine Pivotisierung durchgeführt wird, indem in jedem Iterationsschritt des Householder-Verfahrens die Spalte mit der größten

Norm an die erste Spalte der aktuell betrachteten Submatrix getauscht wird. Diese Spaltenvertauschungen werden in der Permutationsmatrix P gespeichert. In dieser Darstellung ist es möglich, Gleichungssysteme der Form $Ax = y$ numerisch stabil zu lösen, indem zuerst das System $R\tilde{x} = Q^T y$ gelöst wird, was leicht ist, da es sich bei R um eine obere Dreiecksmatrix handelt. Anschließend ergibt sich die Lösung des ursprünglichen Systems durch $x = P\tilde{x}$.

Die konkrete QR-Zerlegung einer Matrix wird durch das Interface *IRealQRFactorization* dargestellt. Das Interface *IRealQRFactorizationAlgorithm* stellt einen Algorithmus dar, der eine QR-Zerlegung durchführt. Die Klasse *RealQRFactorizationAlgorithm* ist die in diesem Programm verwendete Implementierung des Interfaces. Schließlich stellt das Interface *IRealLinearSolverAlgorithm* einen Löser von linearen Gleichungssystemen dar. Die Implementierung dieses Interfaces *DefaultRealLinearSolverAlgorithm* verwendet dafür wie oben beschrieben die QR-Zerlegung. Schließlich existiert noch die statische Klasse *DefaultAlgorithms*, in der die im Programm standardmäßig zu verwendenden Algorithmen gesetzt werden können.

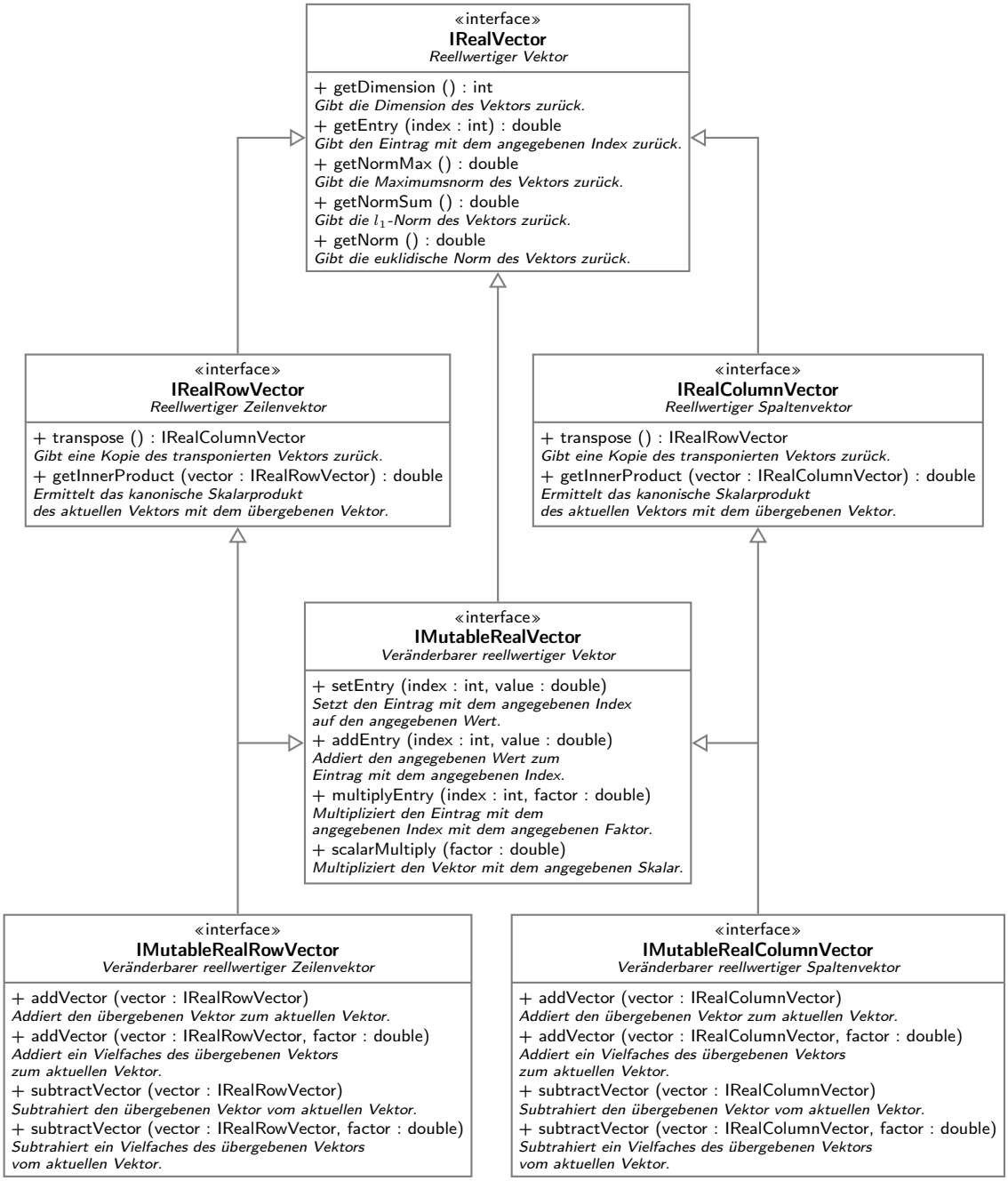

Abbildung 15: Vektor-Interfaces

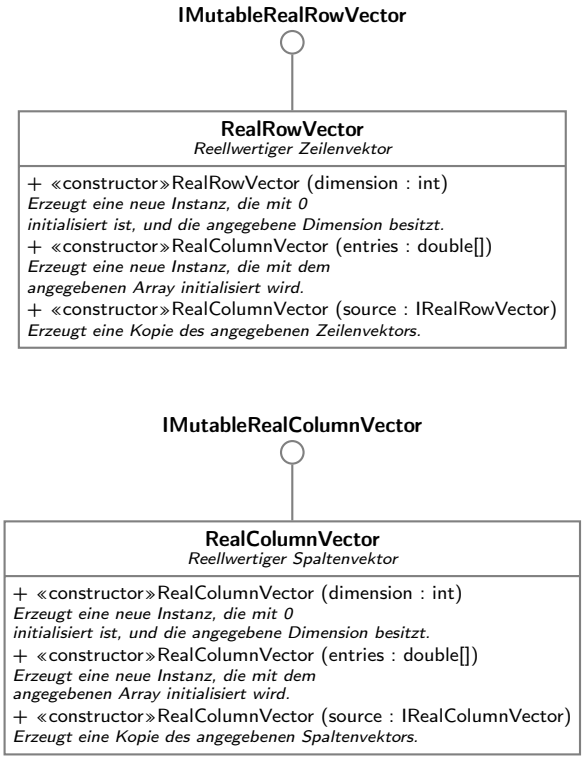

Abbildung 16: Vektor-Klassen

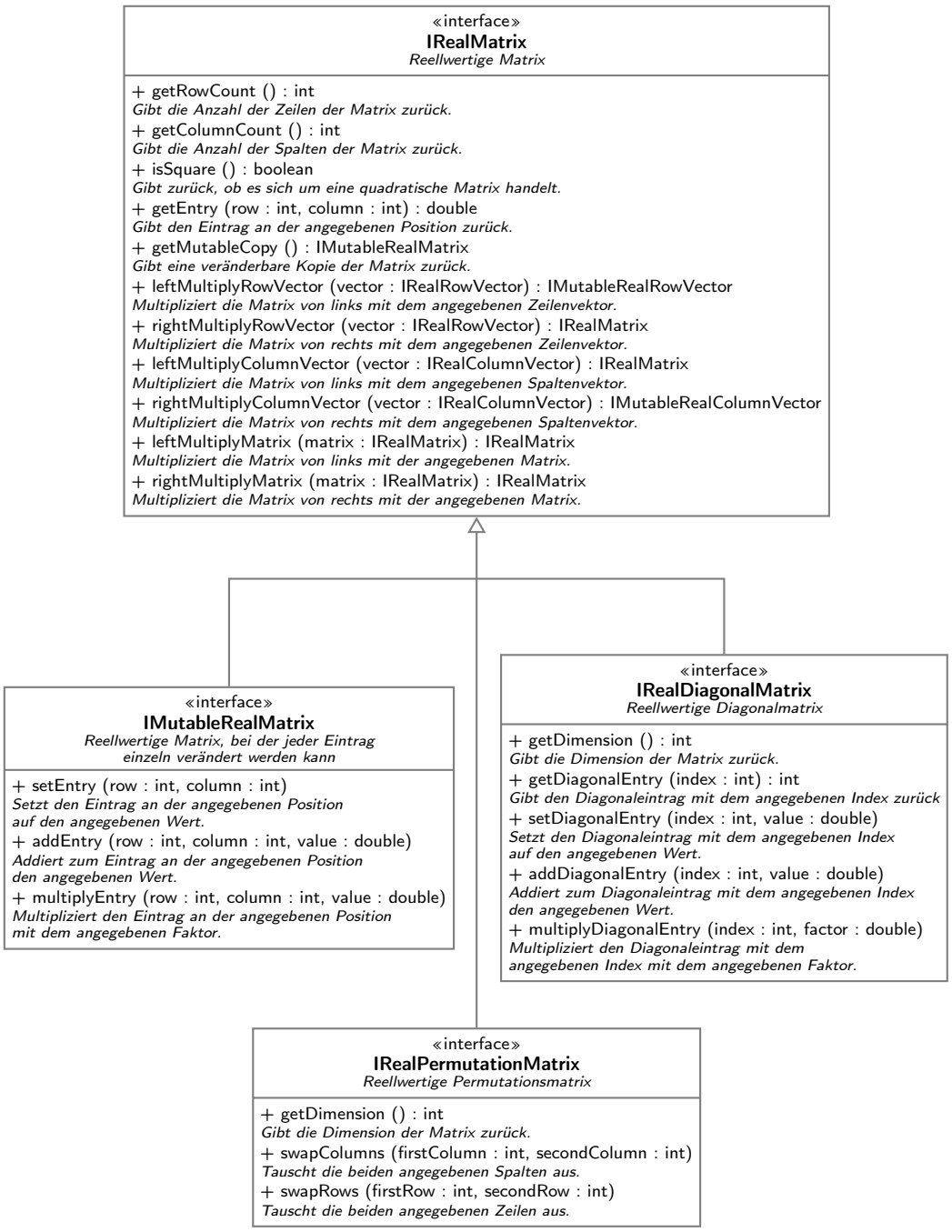

Abbildung 17: Matrix-Interfaces

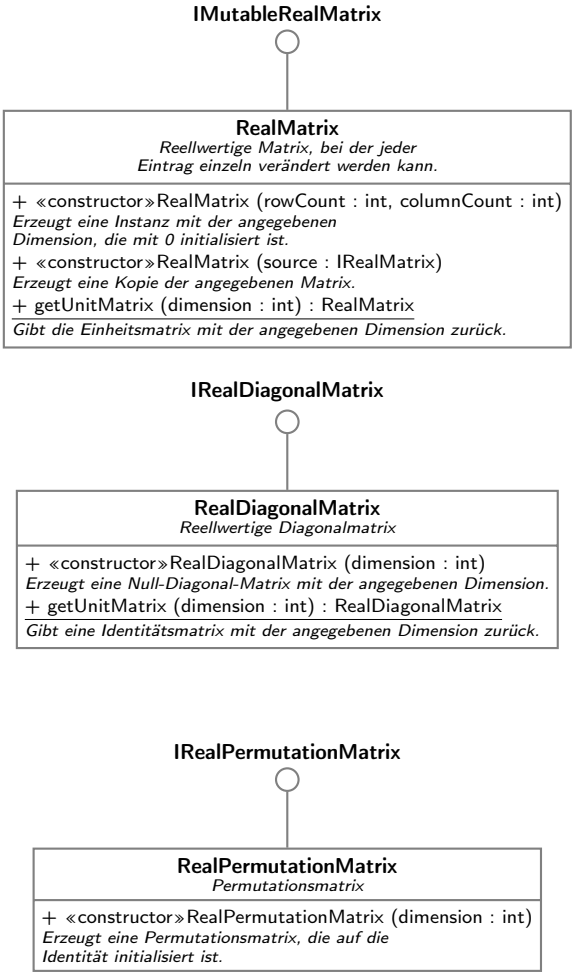

Abbildung 18: Matrix-Klassen

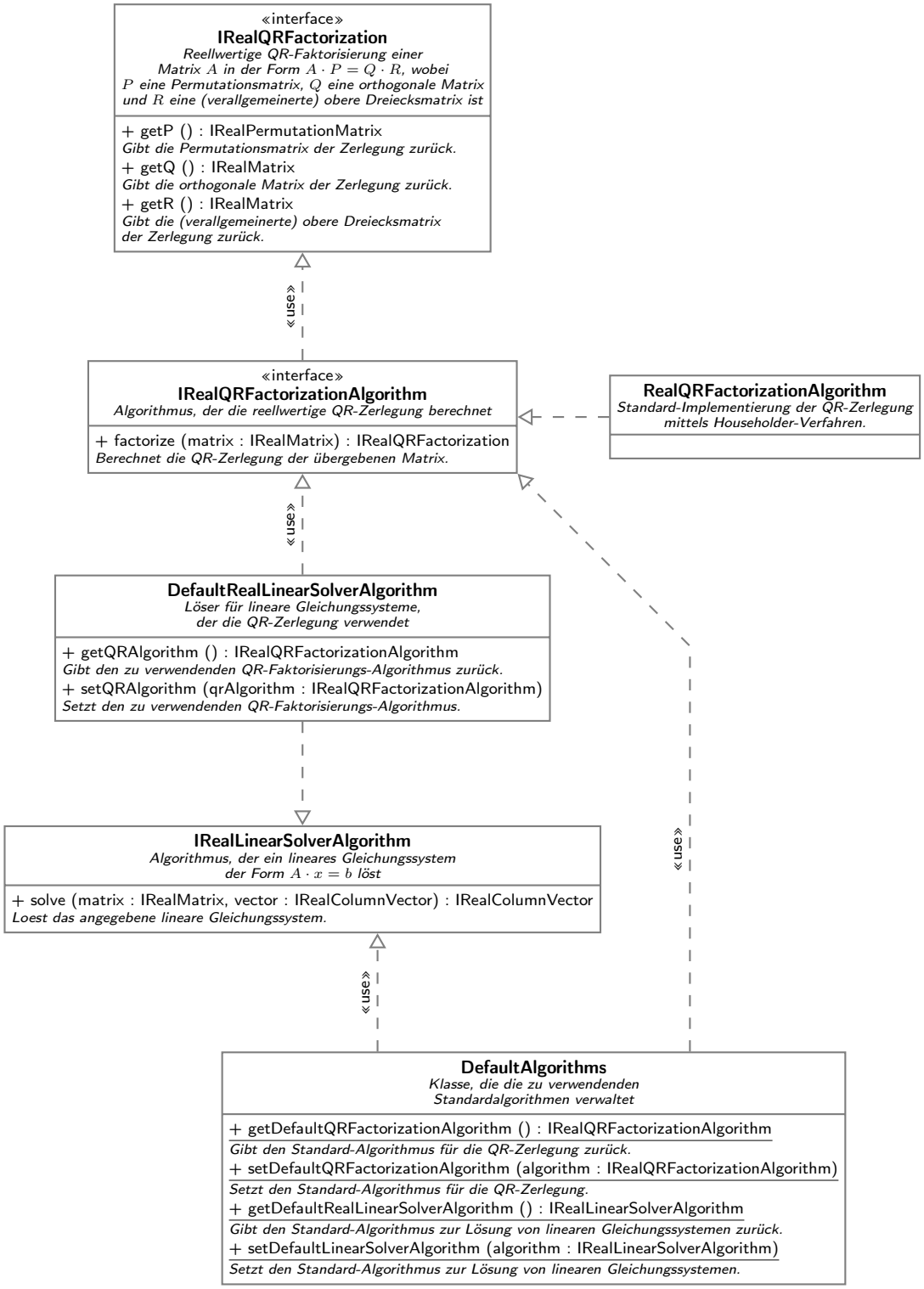

Abbildung 19: QR-Zerlegung und Löser für lineare Gleichnugssysteme

4.4 Exponentielles Boltzmann-Orakel

Die Implementierung des exponentiellen Boltzmann-Orakels und die Interfaces, die als Schnittstelle zu anderen Modulen dienen, sind im Package *at.techmath.boltzmann.oracle* zu finden. In Abbildung 20 sehen wir den prinzipiellen Aufbau. Das Interface *IExponentialBoltzmannOracleFactory* stellt eine Factory dar, die aus kombinatorischen Spezifikationen ein exponentielles Boltzmann-Orakel (*IExponentialBoltzmannOracle*) erzeugen kann. Für ein wohlfundiertes System $\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{H}}\big(\mathcal{Z}, \boldsymbol{\mathcal{Y}}\big)$ mit Lösung $\boldsymbol{\mathcal{S}}$, für das von der Factory ein Orakel erzeugt wurde, wertet die Methode *evaluate* des Orakels die angegebene Komponente des zur Lösung gehörenden Vektors der exponentiellen erzeugenden Funktionen $\mathbf{S}(x)$ an der angegebene Stelle aus.

In Abbildung 21 sehen wir den Aufbau der konkreten Implementierung des *IExponentialBoltzmannOracle*-Interfaces, der Klasse *ExponentialBoltzmannOracle*. Aus der Spezifikation wird in der Konstruktionsphase des Orakels zu jeder benannten Spezies eine korrespondierende Klasse, die die exponentielle erzeugende Funktion der Spezies darstellt, instanziert. Es ergibt sich also eine interne Darstellung des Vektors der multivariaten exponentiellen erzeugenden Funktion $\mathbf{H}(z, \mathbf{y})$. Jede Instanz einer Unterklasse von *ExponentialGeneratingFunction* stellt hierbei eine Komponente dieses Vektors dar. Von dieser Klasse wird einerseits eine Methode *evaluate* veröffentlicht, die die erzeugende Funktion der jeweiligen Spezies an der Stelle $(z, \mathbf{y})$ auswertet, andererseits eine Methode *evaluatePartialDerivative*, die die partielle Ableitung nach der angegebenen $\mathbf{y}$-Komponente an der Stelle $(z, \mathbf{y})$ auswertet.

Es gibt zu jeder Spezies, die in der Spezifikation als benannte Spezies vorkommen kann, eine entsprechende Klasse, die deren erzeugende Funktion repräsentiert. Zusätzlich wird bei den Spezies der endlichen Folgen, der Mengen und der Zyklen unterschieden, ob die Singleton-Spezies oder eine andere benannte Spezies eingesetzt wird.

Die Auswertung der exponentiellen erzeugenden Funktion erfolgt mittels Newton-Iteration, wie sie in Abschnitt 3.4 vorgestellt wird. Es wird also die Iterationsfolge

$$\mathbf{y}^{[0]} = \mathbf{0} \text{ und } \mathbf{y}^{[n+1]} = \mathbf{y}^{[n]} + \Big(\mathbf{Id} - \frac{\boldsymbol{\partial}\mathbf{H}}{\boldsymbol{\partial}\mathbf{Y}}\big(\alpha, \mathbf{y}^{[n]}\big)\Big)^{-1} \cdot \big(\mathbf{H}\big(\alpha, \mathbf{y}^{[n]}\big) - \mathbf{y}^{[n]}\big),$$

berechnet. Dies geschieht unter Verwendung der Funktionen des in Abschnitt 4.3 vorgestellten *at.techmath.boltzmann.linearalgebra*-Packages. Anstatt die Matrix $\big(\mathbf{Id} - \frac{\boldsymbol{\partial}\mathbf{H}}{\boldsymbol{\partial}\mathbf{Y}}\big(\alpha, \mathbf{y}^{[n]}\big)\big)$ zu invertieren, wird das lineare Gleichungssystem

$$\Big(\mathbf{Id} - \frac{\boldsymbol{\partial}\mathbf{H}}{\boldsymbol{\partial}\mathbf{Y}}\big(\alpha, \mathbf{y}^{[n]}\big)\Big) \cdot \mathbf{x} = \mathbf{H}\big(\alpha, \mathbf{y}^{[n]}\big) - \mathbf{y}^{[n]}$$

gelöst. Die Lösung entspricht hier genau dem gewünschten Vektor

$$\big(\mathbf{Id} - \frac{\boldsymbol{\partial}\mathbf{H}}{\boldsymbol{\partial}\mathbf{Y}}\big(\alpha, \mathbf{y}^{[n]}\big)\big) \cdot \big(\mathbf{H}\big(\alpha, \mathbf{y}^{[n]}\big) - \mathbf{y}^{[n]}\big).$$

Da beim Sampling die Stellen, an der die jeweiligen erzeugenden Funktionen ausgewertet werden, konstant sind, wurde weiters ein Cache implementiert, in dem bereits berechnete Werte zur weiteren Verwendung zwischengespeichert werden. Die *evaluate*-Methode der *ExponentialBoltzmannOracle*-Klasse gibt dann, falls der gewünschte Wert schon einmal berechnet wurde, den Wert aus dem Cache zurück, ansonsten wird die Newton-Iteration durchgeführt.

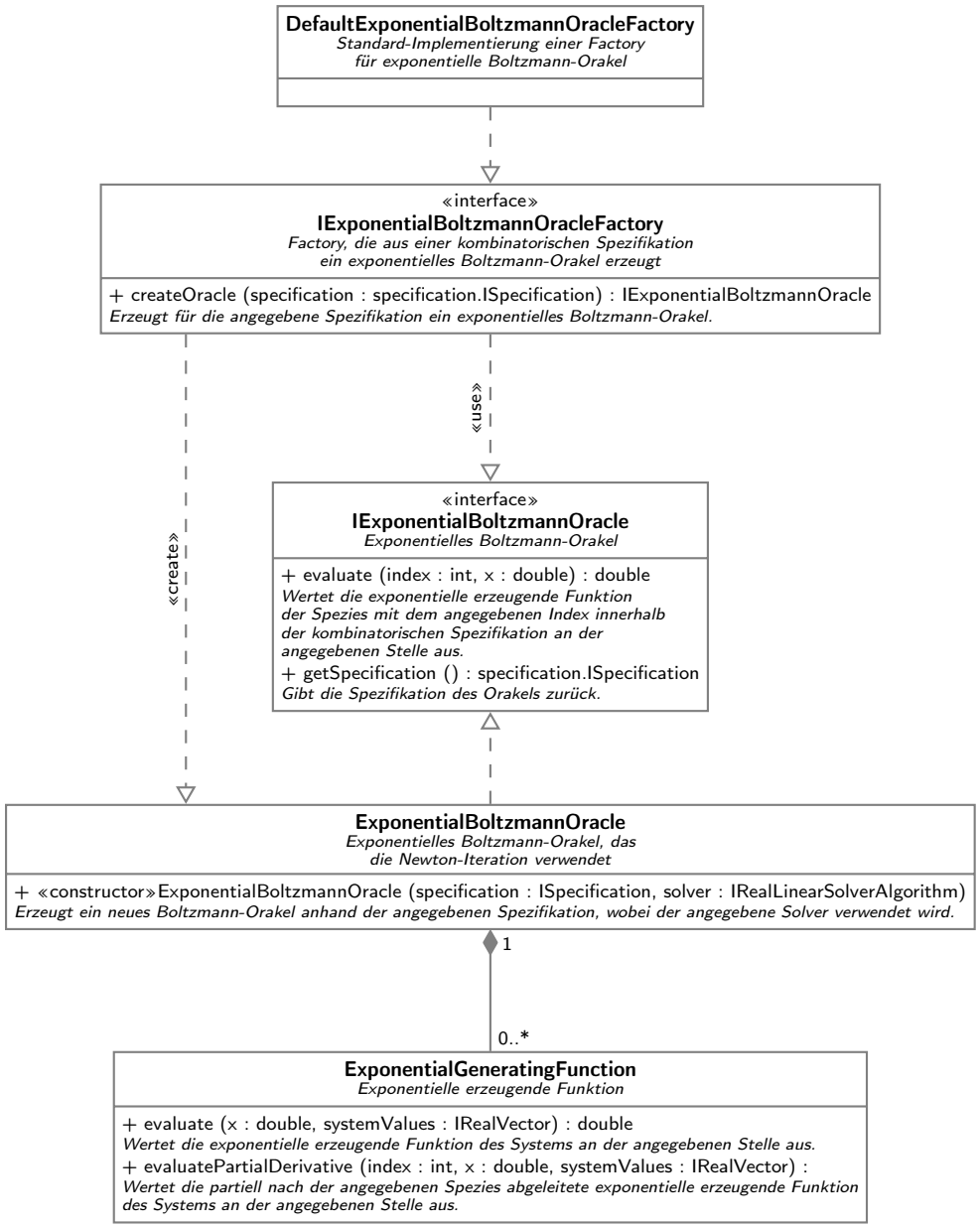

Abbildung 20: Exponentielles Boltzmann-Orakel

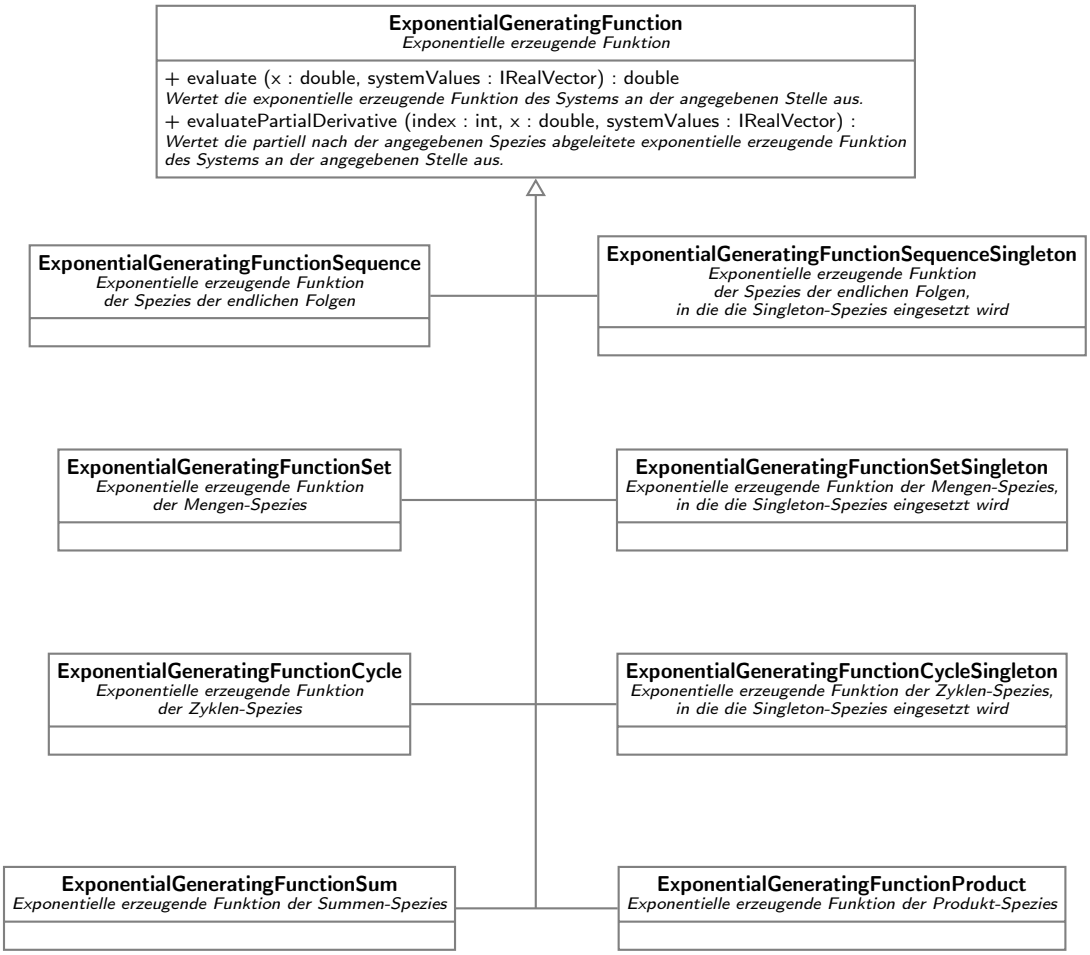

Abbildung 21: Exponentielle erzeugende Funktionen

4.5 Erzeugung von Zufallszahlen

In diesem Abschnitt betrachten wir die Implementierung der Zufallszahlengeneratoren, die für das Sampeln der diversen Spezies benötigt werden. Es wurde für jede Art von Generator eine eigene Methode verwendet, die in den folgenden Unterabschnitten jeweils kurz erklärt werden.

In Abbildung 22 sehen wir die Interfaces, die den anderen Packages zur Verfügung gestellt werden. Die Zufallszahlengenerator-Interfaces treten immer paarweise auf. Zu jedem Generator gibt es eine Factory, die Instanzen des Generators erzeugt. So können bei Bedarf unabhängige Instanzen von Generatoren erzeugt werden und parallel verwendet werden. Außerdem sehen wir in der Abbildung noch, dass die Generatoren für die geometrische, logarithmische und die Poisson-Verteilung auf dem Generator für die Gleichverteilung auf $[0, 1)$ aufbauen.

4.5.1 Erzeugung unabhängig gleichverteilter Zufallszahlen

Zur Erzeugung von unabhängig gleichverteilter Pseudo-Zufallszahlen wird von der Klasse *UniformRandomGeneratorComplementaryMultiplyWithCarry* der in [13, S. 9] angegebene Algorithmus 10 verwendet. Dieser Pseudozufallszahlengenerator hat eine Periodenlänge $> 2^{131086}$, besteht alle für Pseudozufallszahlen verwendete Tests auf Zufälligkeit und ist nach einer einmaligen Initialisierungsphase sehr effizient.

Zur Initialisierung des im Algorithmus verwendeten Seed-Arrays verwendet die Klasse eine Funktion der Java-Klassenbibliothek, mit der Universally Unique Identifier erzeugt werden können (siehe `http://tools.ietf.org/html/rfc4122`). Aus dem erzeugten UUID und der aktuellen Systemzeit wird ein 32-Byte-Array initialisiert, auf dem die SHA-256-Hashfunktion (siehe `http://tools.ietf.org/html/rfc4634`) 512 mal iteriert wird. Das Ergebnis jedes Iterationsschritt wird bei null beginnend in die jeweils acht nächsten Integer des Seed-Arrays geschrieben.

Algorithmus 10 wird von der Methode *getInt* implementiert, wobei hier zu beachten ist, dass das Most Significant Bit als Vorzeichen-Bit interpretiert wird, da Java leider keine Datentypen ohne Vorzeichen besitzt. Alle restlichen Methoden, die vom Interface *IUniformRandomGenerator* definiert werden, bauen auf dieser Methode auf. Ein zufälliger *Long* l wird etwa aus zwei unabhängigen, zufälligen *Integers* i_1 und i_2 erzeugt, wobei einer mit dem Faktor 2^{32} multipliziert wird, und der andere addiert wird. Zur Erzeugung eines *Doubles* im Intervall $[0, 1)$ werden die Bits eines zufälligen *Longs* als Mantisse der Gleitkommazahl interpretiert. Java stellt hier eine Methode zur Verfügung, mit der ein *Long* auf Bitebene in einen IEEE 754-Double konvertiert werden kann.

4.5.2 Erzeugung unabhängig geometrisch verteilter Zufallszahlen

Die in Abbilung 24 verwendete Implementierung des *IGeometricRandomGenerator*-Interfaces, die Klasse *GeometricRandomGeneratorInverseMethod*, verwendet die sogenannte *Inversen-Methode*, um aus auf $[0, 1)$ gleichverteilten Zufallszahlen geometrisch verteilte Zufallszahlen zu erzeugen. Diese Methode wird hier kurz zusammengefasst (siehe zum Beispiel [11, S. 113-116]). Wir beginnen mit einer Definition.

Definition 4.5.1. Sei F eine wahrscheinlichkeitstheoretische Verteilungsfunktion auf $\mathbb{R}$, so heißt die durch

$$F^{-1}(p) := \inf\{x \in \mathbb{R} : p \leq F(x)\} \tag{62}$$

definierte Funktion F^{-1} die *(verallgemeinerte) inverse Verteilungsfunktion.*

Bemerkung. Da Verteilungsfunktionen rechtsstetig sind, können wir in der Definition das Infimum durch ein Minimum ersetzen.

Die Bedeutung für die Zufallszahlenerzeugung wird klar, wenn wir den folgenden in [11, S. 114] bewiesenen Satz betrachten, der die Grundlage dafür bildet, dass wir aus gleichverteilten Zufallszahlen welche erzeugen können, die einer beliebigen gewünschten Verteilung gehorchen.

Algorithmus 10 Complementary Multiply With Carry

Eingabe: Ein in der Initialisierungsphase des Programms auf einen zufälligen Zustand initialisiertes Seed-Array $Q[4096]$, der aktuelle Index in das Seed-Array i, der in der Initialisierungsphase auf 4095 gesetzt wurde, und eine Variable c, die in der Initialisierungsphase mit 123 initialisiert wurde.

Ausgabe: Pseudozufallszahl $\in \{0, 1, \ldots, 2^{32} - 1\}$

Die Variablen t und a sind vom Typ *64 Bit unsigned Integer*, alle anderen Variablen sind vom Typ *32 Bit unsigned Integer*.

$a \leftarrow 18782$
$m \leftarrow 2^{32} - 2$
$i \leftarrow (i + 1) \mod 4096$
$t \leftarrow a \cdot Q[i] + c$
$c \leftarrow t/2^{32}$
$x \leftarrow t + c$
if $x < c$ **then**
  $x \leftarrow x + 1$
  $c \leftarrow c + 1$
end if
$Q[i] \leftarrow m - x$
return $Q[i]$

Satz 4.5.2. *Sei F eine wahrscheinlichkeitstheoretische Verteilungsfunktion auf $\mathbb{R}$ mit der verallgemeinerten Inversen F^{-1} und U eine auf $[0, 1)$ gleichverteilte Zufallsvariable, so gilt, dass $F^{-1} \circ U$ nach F verteilt ist.*

Wir müssen also nur die inverse Verteilungsfunktion berechnen können, und diese an einer auf $[0, 1)$ gleichverteilten Stelle auswerten. Diesen Sachverhalt machen wir uns nun für die Erzeugung einer geometrisch verteilten Zufallszahl zunutze. Sei X nun eine mit Parameter $x \in (0, 1)$ geometrisch verteilte Zufallsvariable, dann gilt für alle $n \in \mathbb{N}_0$

$$P\big(X = n\big) = (1 - x) \cdot x^n.$$

Also folgt für die Verteilungsfunktion für alle $n \in \mathbb{N}_0$ und alle $a \in [0, 1)$

$$F(n + a) = \sum_{k=0}^{n} P\big(X = k\big) = \sum_{k=0}^{n} (1 - x) \cdot x^k = (1 - x) \sum_{k=0}^{n} x^k = (1 - x) \frac{1 - x^{n+1}}{1 - x} = 1 - x^{n+1}.$$

Sei nun $p \in [0, 1)$ beliebig, so gilt weiters für alle $n \in \mathbb{N}_0$

$$F(n) \geq p \;\Leftrightarrow\; 1 - x^{n+1} \geq p \;\Leftrightarrow\; 1 - p \geq x^{n+1} \;\Leftrightarrow$$
$$\ln(1 - p) \geq (n + 1) \ln(x) \;\Leftrightarrow\; \frac{\ln(1 - p)}{\ln(x)} \leq n + 1 \;\Leftrightarrow\; \frac{\ln(1 - p)}{\ln(x)} - 1 \leq n.$$

Daher gilt für die inverse Verteilungsfunktion

$$F^{-1}(p) = \min\big\{x \in \mathbb{R} \,:\, p \leq F(x)\big\} = \min\big\{n \in \mathbb{N}_0 \,:\, p \leq F(n)\big\}$$
$$= \min\Big\{n \in \mathbb{N}_0 \,:\, n \geq \frac{\ln(1 - p)}{\ln(x)} - 1\Big\} = \Big\lceil \frac{\ln(1 - p)}{\ln(x)} - 1 \Big\rceil.$$

Diese Funktion wird in der Methode *generateGeometric* der Klasse *GeometricRandomGeneratorInverseMethod* ausgewertet.

4.5.3 Erzeugung unabhängig Poisson-verteilter Zufallszahlen

Zur Implementierung eines Generators für Poisson-verteilte Zufallszahlen, deren Struktur in Abbildung 25 dargestellt ist, machen wir uns zunutze, dass wir mit der Inversenmethode sehr einfach

exponentialverteilte Zufallsvariablen erzeugen können, sowie, dass die Exponentialverteilung und die Poissonverteilung wie folgt beschrieben zusammenhängen.

Definition 4.5.3. Eine Zufallsvariable X mit Dichte $f_\lambda(x) := \lambda \cdot \exp(-\lambda x) \cdot \mathbb{1}_{[0,\infty)(x)}$ heißt *exponentialverteilt mit Parameter* λ. Wir schreiben dann auch $X \sim \mathrm{Ex}_\lambda$. Weiters heißt eine Zufallsvariable X mit Dichte $f_{\lambda,n} := \exp(-\lambda x) \cdot \frac{\lambda^n x^{n-1}}{(n-1)!} \cdot \mathbb{1}_{[0,\infty)}(x)$ *erlangverteilt mit n Freiheitsgraden und Parameter* λ. Die Exponentialverteilung ist also eine Erlangverteilung mit einem Freiheitsgrad.

Satz 4.5.4. *Seien $X_1, \ldots, X_n$ unabhängig mit Parameter λ exponentialverteilte Zufallsvariablen, dann ist die Summe $\sum_{k=1}^n X_k$ mit n Freiheitsgraden und Parameter λ erlangverteilt.*

Beweis. Die Dichte der Summe $\sum_{k=1}^n X_k$ bezeichnen wir mit $g(x)$, die Dichte von X_k mit $f_k(x)$. Der Beweis erfolgt durch vollständige Induktion. Für $n = 1$ ist die Aussage erfüllt, da die Erlangvereiltung mit einem Freiheitsgrad genau die Exponentialverteilung ist. Gelte die Aussage nun für ein beliebiges $n \in \mathbb{N}$, dann sehen wir, da die Dichte von unabhängigen Zufallsvariablen gleich der Faltung der Einzeldichten ist, dass für $x \in \mathbb{R}$ gilt, dass

$$\begin{aligned}
g_n(x) &= \int_{-\infty}^{\infty} g_{n-1}(t) \cdot f_n(x-t)\, dt \\
&= \int_{-\infty}^{\infty} \exp(-\lambda t) \cdot \frac{\lambda^{n-1} t^{n-2}}{(n-2)!} \cdot \mathbb{1}_{[0,\infty)}(t) \cdot \lambda \cdot \exp(-\lambda(x-t)) \cdot \mathbb{1}_{[0,\infty)}(x-t)\, dt \\
&= \int_0^x \exp(-\lambda t) \cdot \frac{\lambda^{n-1} t^{n-2}}{(n-2)!} \cdot \lambda \cdot \exp(-\lambda(x-t))\, dt \cdot \mathbb{1}_{[0,\infty)}(x) \\
&= \exp(-\lambda x) \frac{\lambda^n}{(n-2)!} \int_0^x t^{n-2}\, dt \cdot \mathbb{1}_{[0,\infty)}(x) = \exp(-\lambda x) \frac{\lambda^n x^{n-1}}{(n-1)!} \cdot \mathbb{1}_{[0,\infty)}(x).
\end{aligned}$$

□

Definieren wir nun $S_n := \sum_{k=0}^n X_k$ für alle $n > 0$ und $S_0 := 0$, so sehen wir mittels partieller Integration, dass

$$\begin{aligned}
P\big(S_n > 1\big) &= \int_1^\infty \exp(-\lambda t) \cdot \frac{\lambda^n t^{n-1}}{(n-1)!}\, dt \\
&= -\frac{1}{\lambda} \exp(-\lambda t) \cdot \frac{\lambda^n t^{n-1}}{(n-1)!}\bigg|_1^\infty + \int_1^\infty \frac{1}{\lambda} \exp(-\lambda t) \cdot \frac{\lambda^n (n-1) t^{n-2}}{(n-1)!}\, dt \\
&= \exp(-\lambda) \frac{\lambda^{n-1}}{(n-1)!} + \int_1^\infty \exp(-\lambda t) \frac{\lambda^{n-1} t^{n-2}}{(n-2)!}\, dt \\
&= \exp(-\lambda) \frac{(\lambda)^{n-1}}{(n-1)!} + P\big(S_{n-1} > 1\big) \text{ für } n > 1, \text{ und} \\
P\big(S_1 > 1\big) &= \int_1^\infty \lambda \exp(-\lambda t)\, dt = -\exp(-\lambda t)\big|_1^\infty + 0 = \exp(-\lambda) \frac{\lambda^0}{0!} + P\big(S_0 > 1\big).
\end{aligned}$$

Weil das Ereignis $\big[S_{n+1} > 1\big] \subseteq \big[S_n > 1\big]$ ist, gilt also für alle $n \in \mathbb{N}_0$, dass

$$P\big(N = n\big) = P\big(S_n \le 1 < S_{n+1}\big) = P\big(S_{n+1} > 1\big) - P\big(S_n > 1\big) = \exp(-\lambda) \frac{\lambda^n}{n!},$$

wobei $N := \max\big\{n \in \mathbb{N}_0 : S_n \le 1\big\} = \min\big\{n \in \mathbb{N} : S_n > 1\big\} - 1$. Die Zufallsvariable N ist daher Poisson-verteilt mit Parameter λ. Die Verteilungsfunktion $F(x)$ der Exponentialverteilung mit Parameter λ ist gegeben durch

$$\begin{aligned}
F(x) &= \int_{-\infty}^x \lambda \exp(-\lambda t) \cdot \mathbb{1}_{[0,\infty)}(t)\, dt = \begin{cases} \int_0^x \lambda \exp(-\lambda t)\, dt & \text{für } x \ge 0, \\ 0 & \text{sonst,} \end{cases} \\
&= \begin{cases} \big[-\exp(-\lambda t)\big]_0^x & \text{für } x \ge 0, \\ 0 & \text{sonst,} \end{cases} = \begin{cases} 1 - \exp(-\lambda x) & \text{für } x \ge 0, \\ 0 & \text{sonst,} \end{cases} \\
&= \big(1 - \exp(-\lambda x)\big) \cdot \mathbb{1}_{[0,\infty)}(x).
\end{aligned}$$

Sei nun $p \in (0,1)$, dann gilt für alle $x \in \mathbb{R}$

$$F(x) \geq p \;\Leftrightarrow\; \big(1-\exp(-\lambda x)\big)\cdot \mathbb{1}_{[0,\infty)}(x) \geq p \;\Leftrightarrow\; 1-\exp(-\lambda x) \geq p$$
$$\Leftrightarrow\; \exp(-\lambda x) \leq 1-p \;\Leftrightarrow\; -\lambda x \leq \ln(1-p) \;\Leftrightarrow\; x \geq -\frac{\ln(1-p)}{\lambda}.$$

Für die inverse Verteilungsfunktion folgt daher

$$F^{-1}(p) = \min\big\{x \in \mathbb{R} \,:\, F(x) \geq p\big\} = \min\Big\{x \in \mathbb{R} \,:\, x \geq -\frac{\ln(1-p)}{\lambda}\Big\} = -\frac{\ln(1-p)}{\lambda}.$$

Für eine Folge $\big(U_n\big)_{n\in\mathbb{N}}$ von unabhängig auf $[0,1)$ gleichverteilten Zufallsvariablen ist daher die Folge $X_n := -\frac{\ln(1-U_n)}{\lambda}$ unabhängig mit Parameter λ exponentialverteilt, und die Zufallsvariable $\min\big\{n \in \mathbb{N} \,:\, \sum_{k=1}^{n} -\frac{\ln(1-U_k)}{\lambda} > 1\big\} - 1 = \min\big\{n \in \mathbb{N} \,:\, \sum_{k=1}^{n} -\ln(1-U_k) > \lambda\big\} - 1$ Poisson-verteilt mit Parameter λ. Dieses Minimum wird im Algorithmus, der in der Methode *generatePoisson* der Klasse *PoissonRandomGeneratorPoissonProcess* implementiert ist, mittels einer Schleife ausgewertet. Die Anzahl der Schleifendurchläufe entspricht dann dem gewünschten Minimum plus eins.

4.5.4 Erzeugung logarithmisch verteilter Zufallszahlen

Die Implementierung eines Generators für logarithmisch verteilte Zufallszahlen verwendet Algorithmus 11, der in [10, S. 251] zu finden ist, und aus zwei unabhängigen auf $[0,1)$ gleichverteilten Zufallszahlen eine logarithmisch verteilte Zufallszahl erzeugt.

Algorithmus 11 LB - Erzeugung logarithmisch verteilter Zufallszahlen

Eingabe: Unabhängige auf $[0,1)$ gleichverteilte Zufallszahlen u_1, u_2 und Parameter α der logarithmischen Verteilung.

Ausgabe: Mit Parameter α logarithmisch verteilte Zufallszahl.

$h \leftarrow \ln(1-\alpha)$

return $\big\lfloor 1 + \frac{\ln(u_2)}{\ln(1-\exp(u_1 \cdot h)}\big\rfloor$

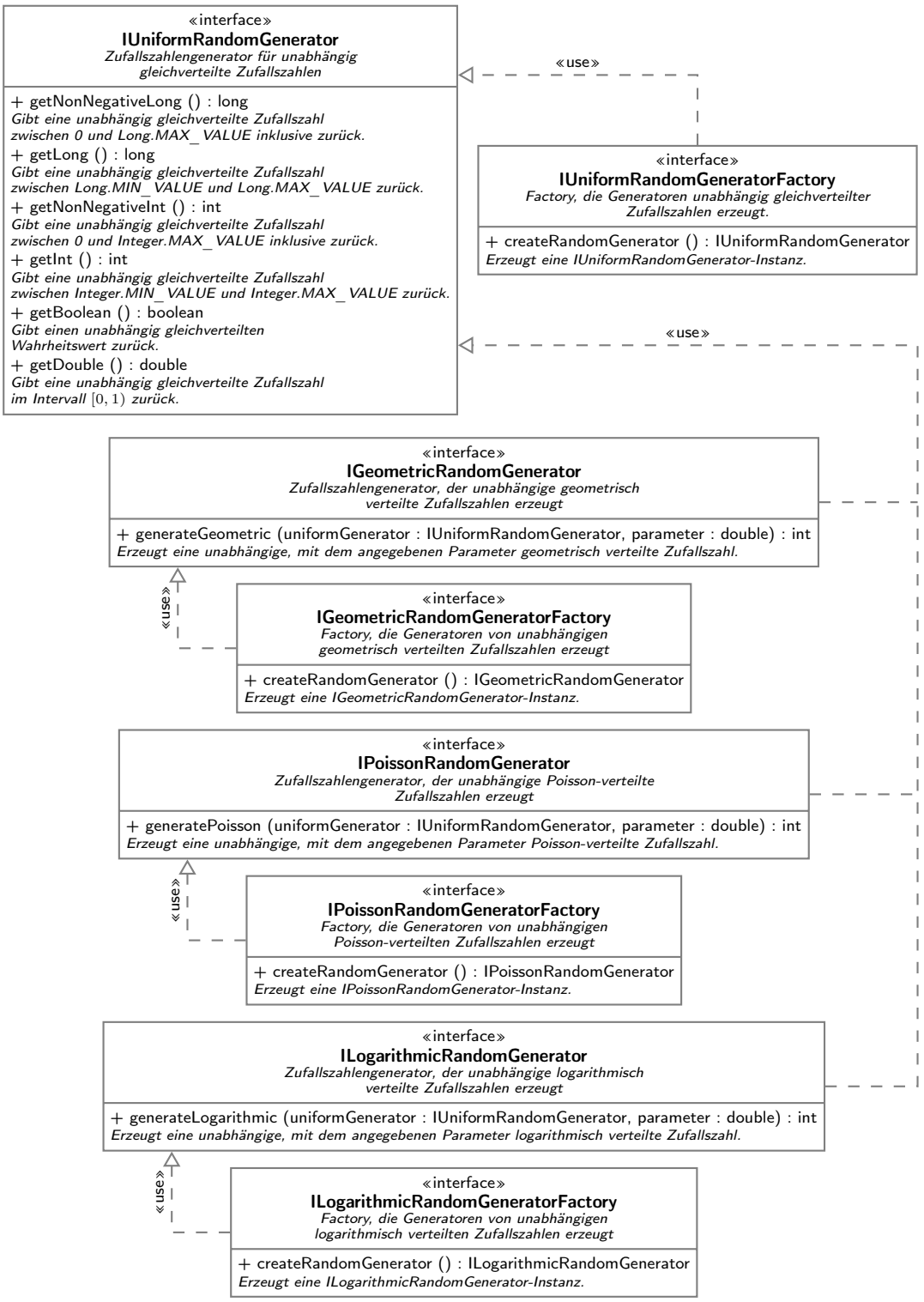

Abbildung 22: Zufallszahlengenerator-Interfaces

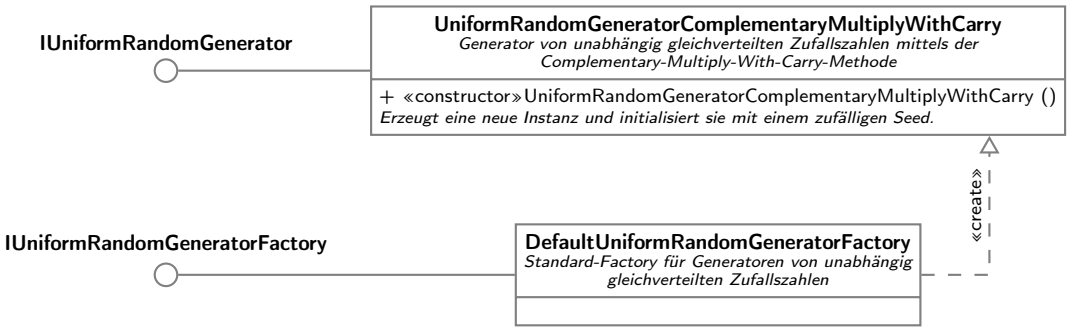

Abbildung 23: Generatoren unabhängig gleichverteilter Zufallszahlen

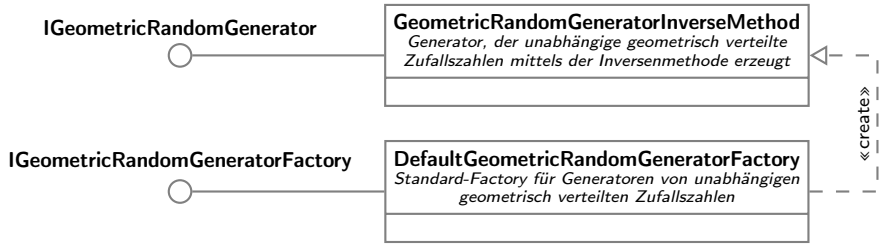

Abbildung 24: Generatoren unabhängig geometrisch verteilter Zufallszahlen

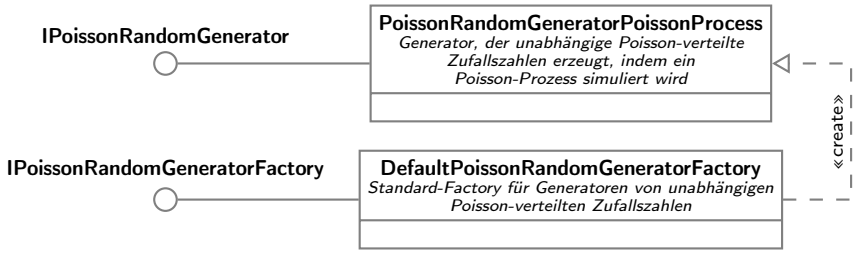

Abbildung 25: Generatoren unabhängig Poisson-verteilter Zufallszahlen

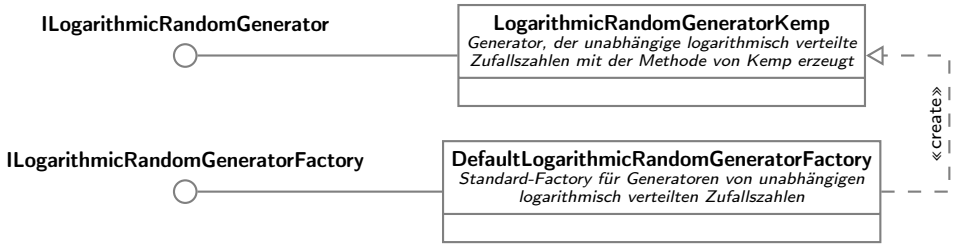

Abbildung 26: Generatoren unabhängig logarithmisch verteilter Zufallszahlen

4.6 Exponentieller Boltzmann-Sampler

Dieser Abschnitt ist der Implementierung des exponentiellen Boltzmann-Samplers gewidmet. Die hier vorgestellten Klassen und Interfaces befinden sich im Package *at.techmath.boltzmann.sampler*. Zuerst werden wir die Interfaces betrachten, die die gesampelten Strukturen repräsentieren. Diese sind in Abbildung 27 dargestellt. Hier gibt es zu jeder Spezies, die in der Spezifikation definiert werden kann, ein korrespondierendes Interface, das eine Struktur der Spezies repräsentiert. Die Strukturen zu den benannten Spezies enthalten jeweils Referenzen auf die untergeordneten Strukturen, die Singleton-Strukturen enthalten das Atom, das auf sie verteilt wurde.

Die Möglichkeit, Strukturen mit den Marken, die zu einer kombinatorischen Spezifikation angegeben werden können, zu versehen, wird durch die in Abbildung 28 dargestellten Interfaces und Klassen geschaffen. Eine Struktur, die mit einer Marke versehen ist, implementiert das *ILabeled-Structure*-Interface. Dieses Interface referenziert eine Instanz des *ILabel*-Interfaces, das wiederum eine konkrete Marke darstellt. Die Unterinterfaces *IIntegerLabel* und *IStringLabel* repräsentieren jeweils eine ganzzahlige beziehungsweise eine Textmarke. Die Implementierungen dieser Interfaces, die Klassen *IntegerLabel* und *StringLabel*, werden von der *createFromSpecificationLabel*-Methode der *Label*-Klasse aus einer Label-Spezifikation erzeugt.

Die konkreten Implementierungen der Struktur-Interfaces sind in den Abbildungen 29 bis 36 dargestellt. Es gibt zu jedem Struktur-Typ jeweils eine Implementierung ohne und eine Implementierung mit Label. Die Klasse, die die markierte Struktur repräsentiert, hat im Klassennamen den Präfix *Labeled* vorangestellt.

Die Schnittstelle des Packages nach aussen ist in Abbildung 38 dargstellt. Das Interface *IExponentialSamplerFactory* stellt eine Factory für exponentielle Boltzmann-Sampler (*IExponentialSampler*) dar. Diese Factory kann aus einer Spezifikation und einer Factory für die Grundmenge einen Sampler erstellen. Der Sampler wiederum stellt die Möglichkeit zur Verfügung, eine Struktur der Spezies, die durch die Spezifikation angegeben ist, zu erzeugen. Weiters ist die Möglichkeit gegeben, eine maximale Strukturgröse anzugeben. Sobald während des Sampling-Prozesses erkannt wird, dass die aktuelle erzeugte Struktur die Schranke für die Strukturgröße überschreitet, wird der Sampling-Vorgang mit einer Exception abgebrochen.

Das Ergebnis des Sampling-Vorgangs wird durch das *ISamplingResult*-Interface dargestellt. Es besteht aus der Struktur und ihrer Größe. Das *IAtomFactory*-Interface beschreibt eine Factory für die Grundmenge, auf der die Strukturen erzeugt werden sollen. Sie stellt eine Methode zur Verfügung, mit der eine Grundmenge der gewünschten Größe in einem Array erstellt werden kann. Wird der Sampler-Factory keine Factory für die Atome übergeben, so wird implizit eine Factory verwendet, die die Menge der natürlichen Zahlen erzeugt.

In Abbildung 39 sehen wir die konkreten Implementierungen der soeben beschriebenen Interfaces. Die Klasse *ExponentialSamplerFactory* ist die Standard-Implementierung der Factory für exponentielle Boltzmann-Sampler. Es können zusätzlich zur Spezifikation und Atom-Factory etliche zu verwendenden Zufallszahlengeneratoren und das zu verwendende Boltzmann-Orakel konfiguriert werden. Diese Factory erstellt aus der angegebenen kombinatorischen Spezifikation eine *ExponentialSampler*-Instanz. Diese Klasse implementiert das *IExponentialSampler*-Interface und ist für den eigentlichen Sampling-Prozess verantwortlich. Korrespondierend mit dem in der Spezifikation angegebenen Gleichungssystem wird für jede benannte Spezies eine *ExponentialSpeciesSampler*-Instanz erzeugt, die Strukturen dieser Spezies sampeln kann. In den Abbildungen 29 bis 36 sehen wir die Unterklassen dieser Klasse, die die entsprechende Spezies sampeln können.

Der Sampling-Prozess läuft in zwei Schritten ab.

1. Erzeugung der Struktur, wobei die Singleton-Strukturen einstweilen nicht mit einem Atom der Grundmenge versehen werden, bei gleichzeitiger Ermittlung der Größe der Struktur.

2. Versehen der Singleton-Unterstrukturen der gesampelten Struktur mit den Atomen der Grundmenge, derart, dass jede mögliche Verteilung der Atome auf die Singleton-Strukturen gleich wahrscheinlich ist.

Der erste Schritt wird von den *ExponentialSpeciesSampler*-Instanzen in Zusammenarbeit mit den entsprechenden *SamplerMethod*-Instanzen durchgeführt. Die *SamplerMethod*-Unterklassen stellen hierbei die Implementierung der Algorithmen, die in Abschnitt 3.2 vorgestellt wurden, dar. Der Grund, warum diese Algorithmen nicht direkt als Methoden der *ExponentialSpeciesSampler*-Klassen implementiert wurden, die sich gegenseitig rekursiv aufrufen, ist, dass die erzeugten Strukturen so groß werden können, dass es dadurch zu einem Stack-Überlauf kommen könnte. Daher wird in dieser Implementierung ein Stack mittels der *SamplingContext*-Klasse selber verwaltet, und die Rekursion in eine Schleife umformuliert.

Für den zweiten Schritt sind die in Abbildung 37 dargestellten Klassen verantwortlich. Die Klassen verteilen die Atome der Grundmenge der natürlichen Zahlen auf die Platzhalter in den Singleton-Unterstrukturen der im ersten Schritt erzeugten Struktur. Auch hier werden die rekursiven Aufrufe aufgrund des selben Arguments in eine Schleife mit eigens verwaltetem Stack umformuliert. Die Zufälligkeit der Verteilung der Atome wird mit dem sogenannten *Swapping*-Algorithmus erzeugt. Hierbei wird das nächste zu verteilende Atom aus einem Array ermittelt, indem ein gleichverteilter Index generiert wird. Das Element an dieser Position wird mit dem aktuell letzten Element des Arrays vertauscht und dann für die Verteilung verwendet. Das neue Rest-Array, das beim jeweils nächsten Schritt verwendet wird, ist das Teilarray des aktuellen Arrays, das aus allen Elementen bis auf das letzte Element besteht.

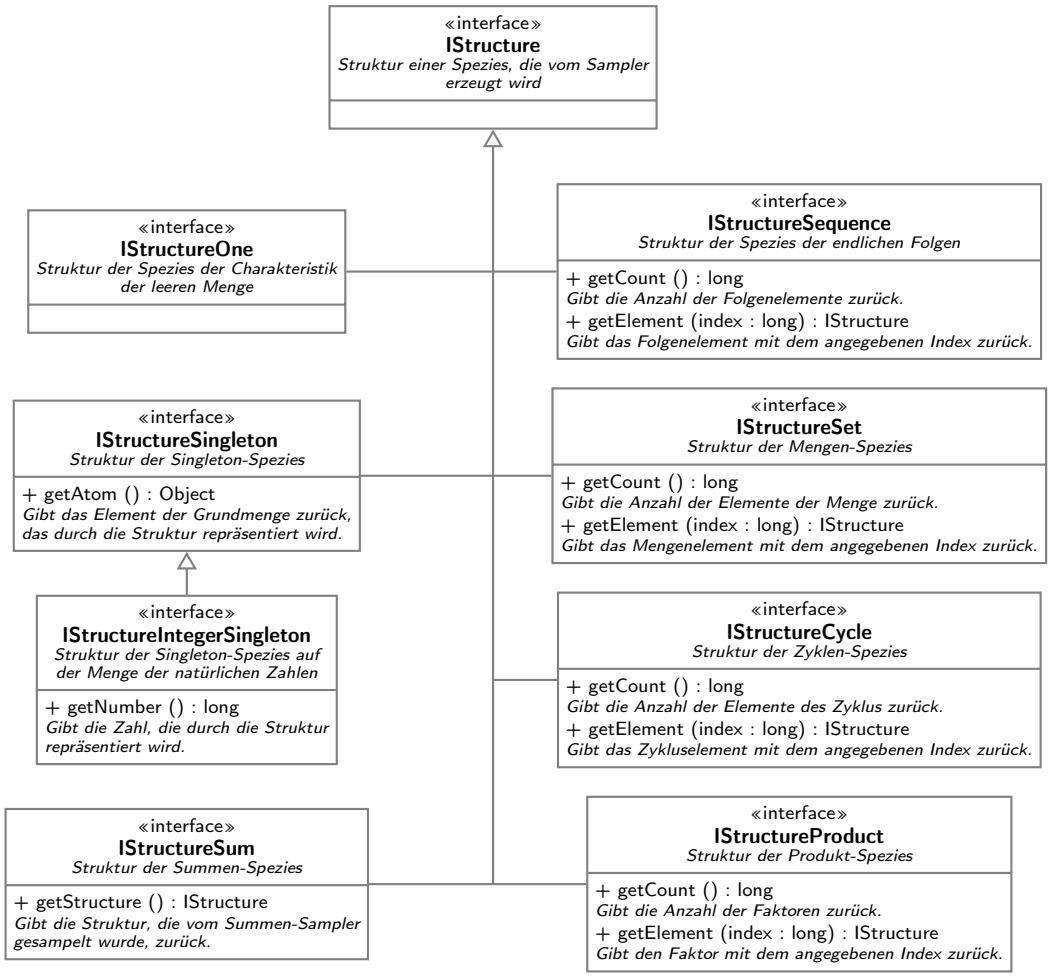

Abbildung 27: Struktur-Interfaces

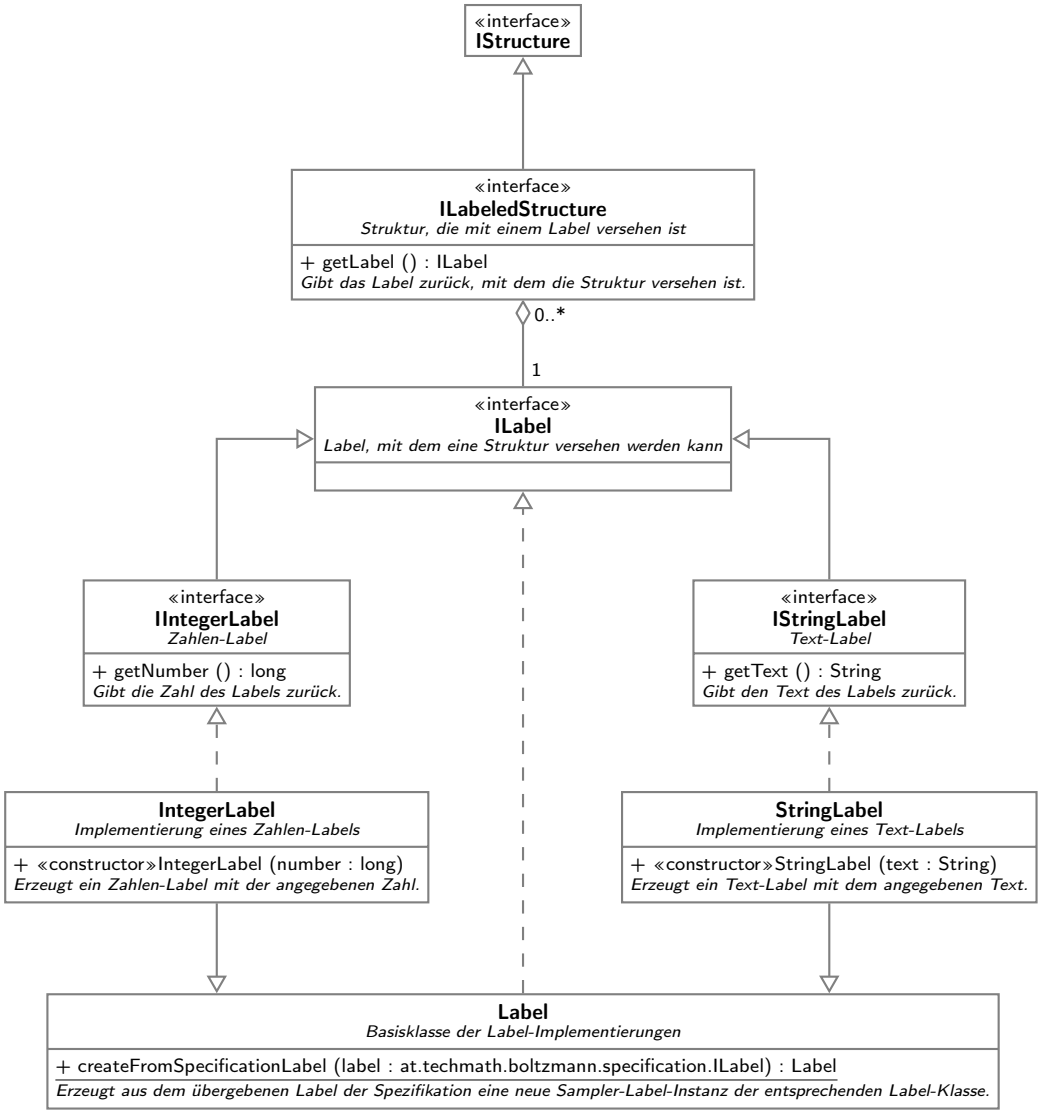

Abbildung 28: Strukturen und Labels

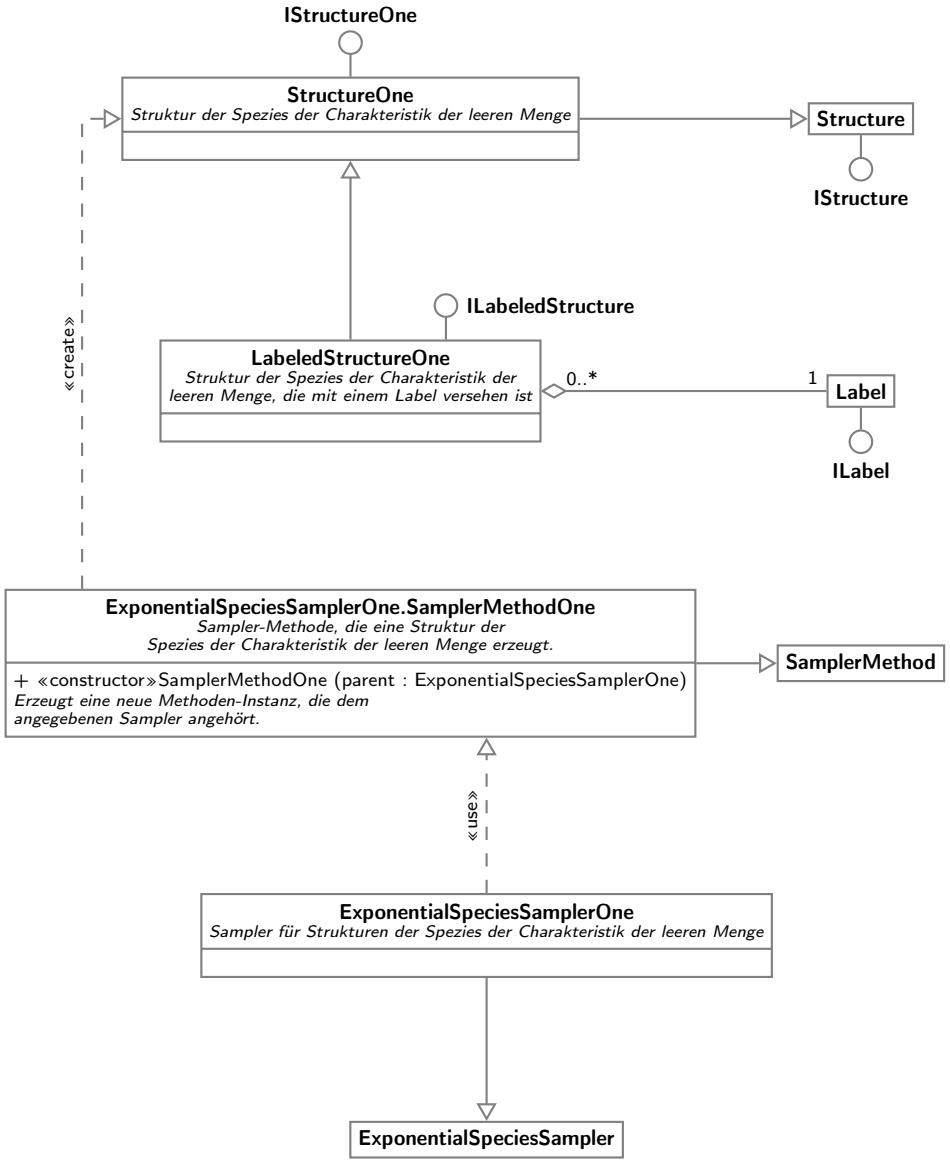

Abbildung 29: Sampler und Strukturen der Spezies der Charakteristik der leeren Menge

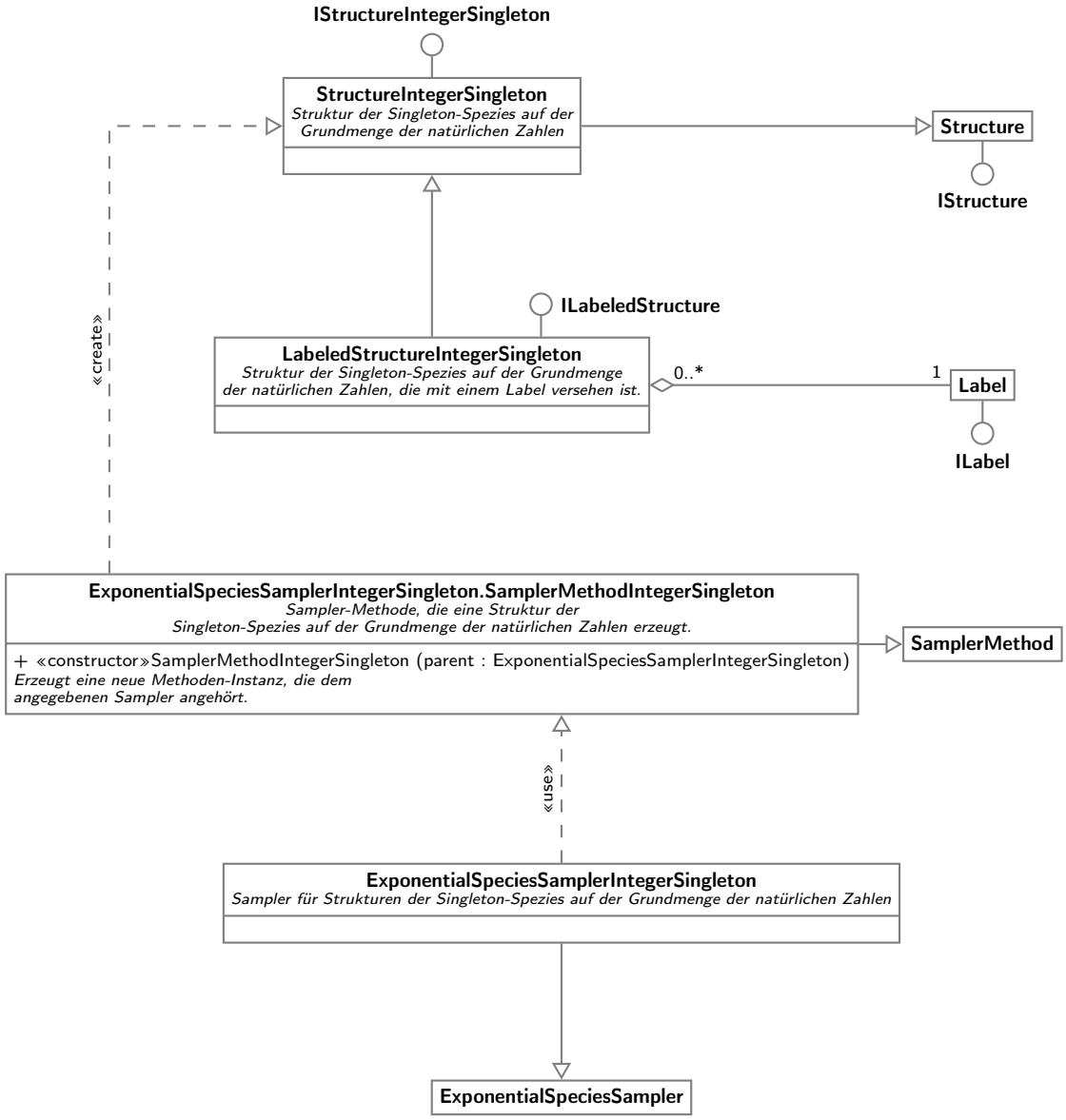

Abbildung 30: Sampler und Strukturen der Singleton-Spezies auf der Grundmenge der natürlichen Zahlen

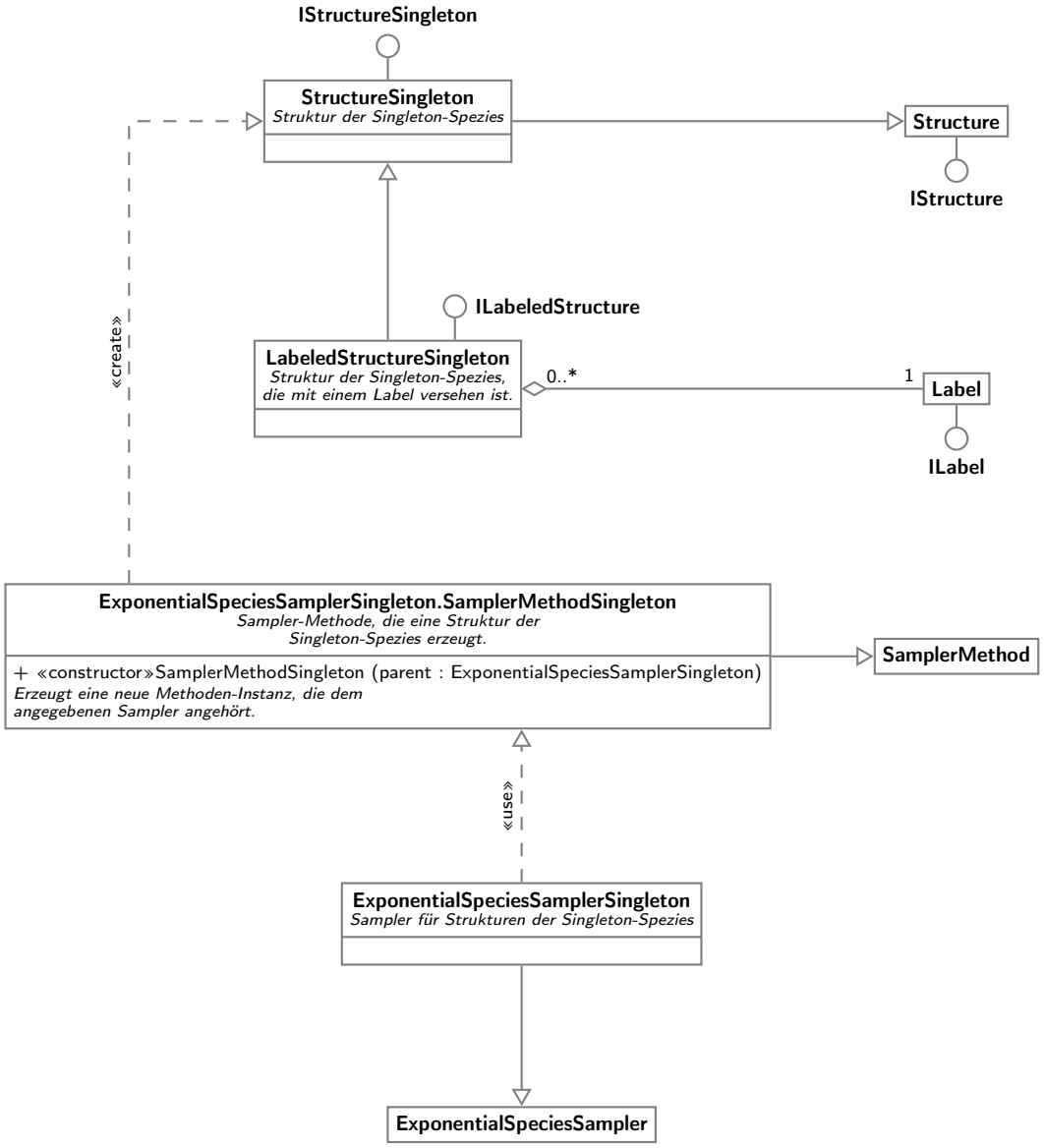

Abbildung 31: Sampler und Strukturen der Singleton-Spezies

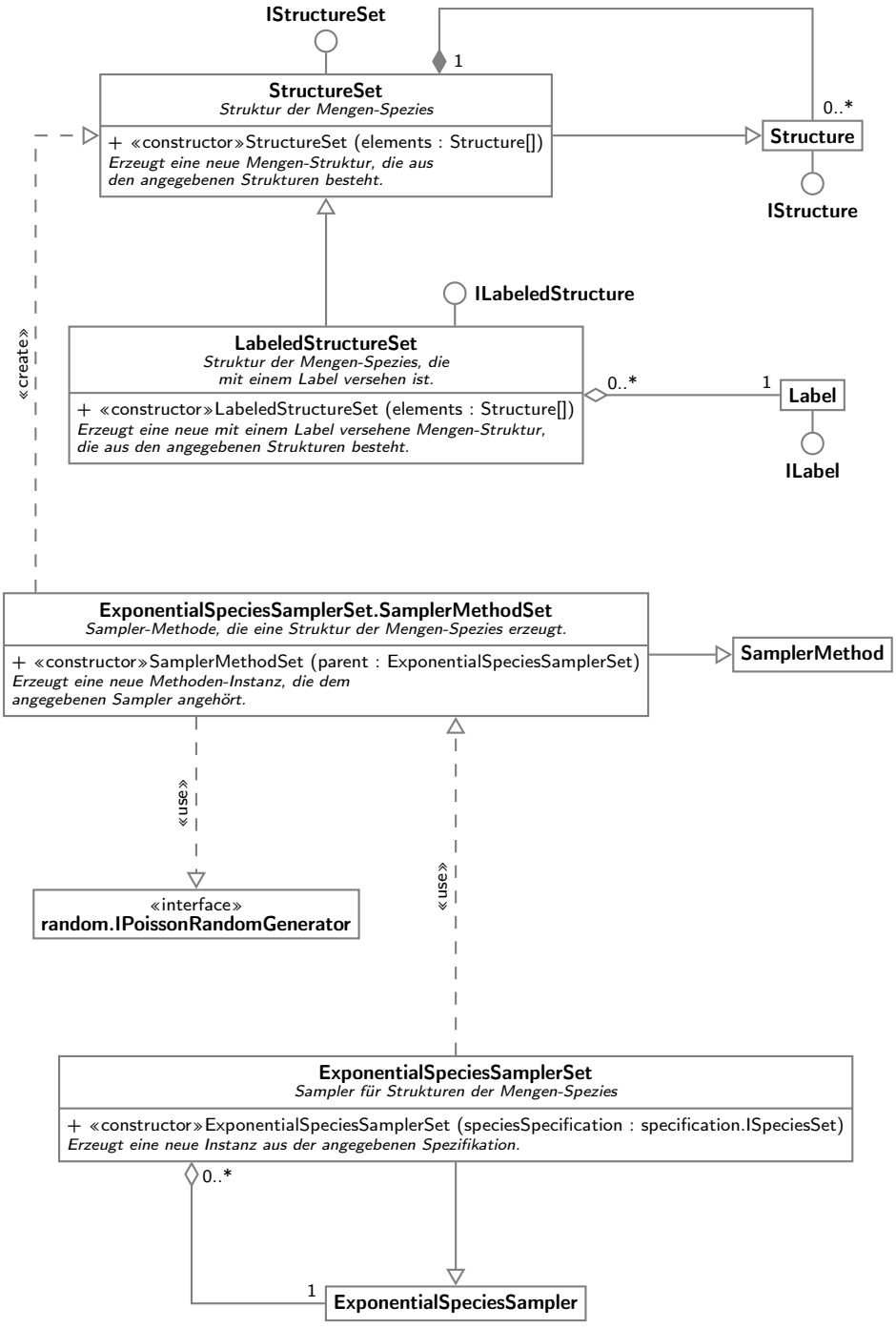

Abbildung 32: Sampler und Strukturen der Mengen-Spezies

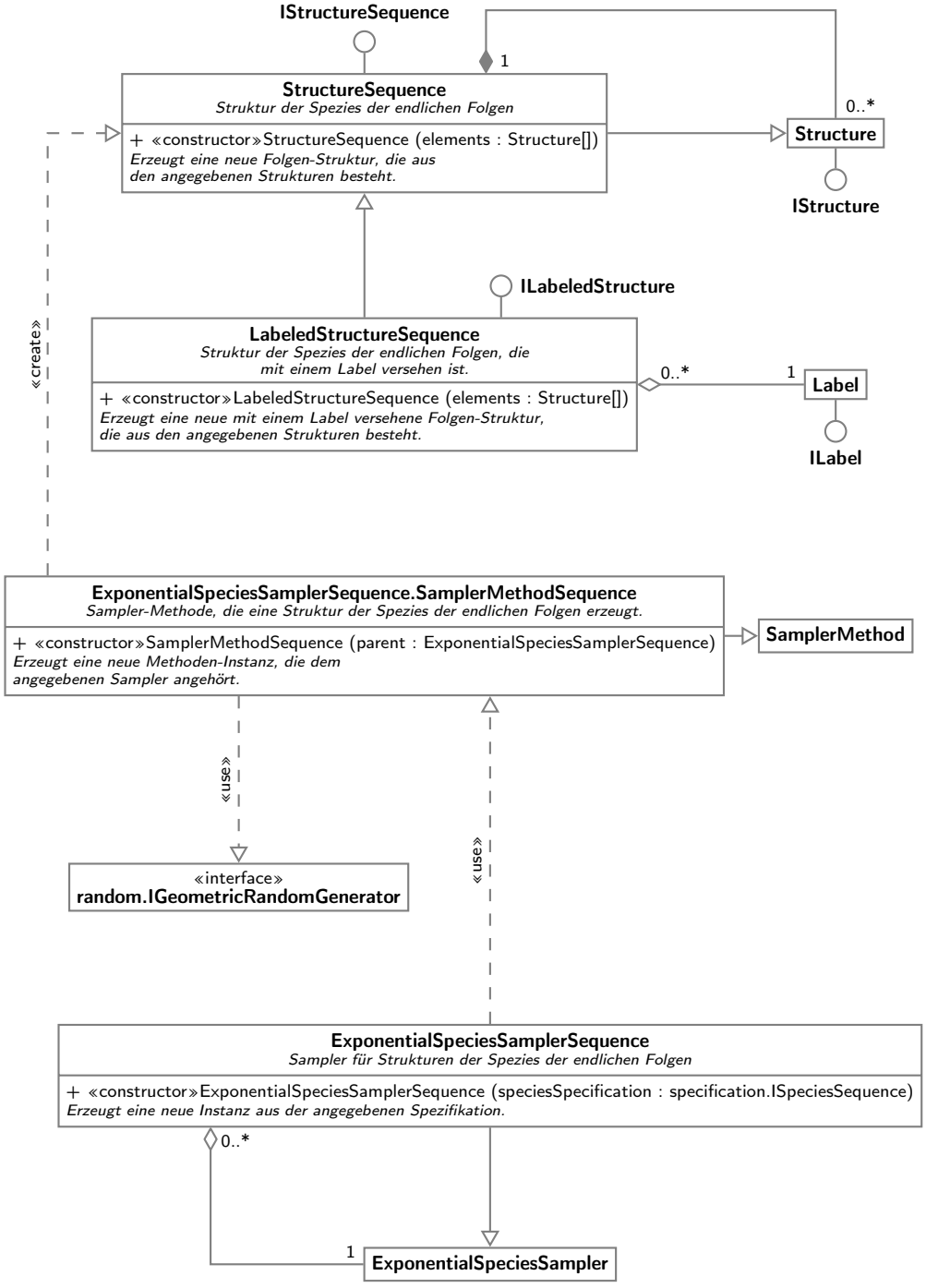

Abbildung 33: Sampler und Strukturen der Spezies der endlichen Folgen

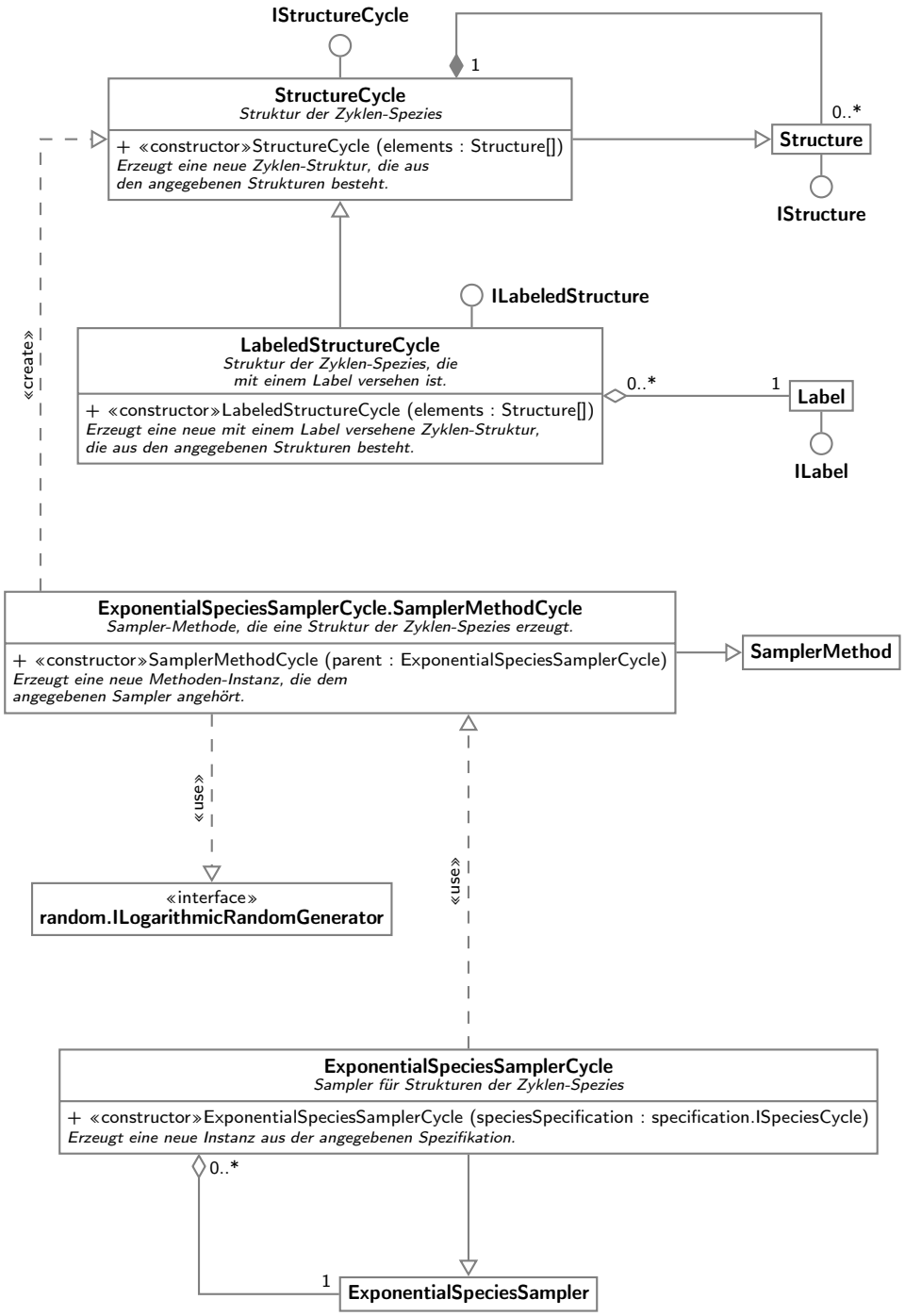

Abbildung 34: Sampler und Strukturen der Zyklen-Spezies

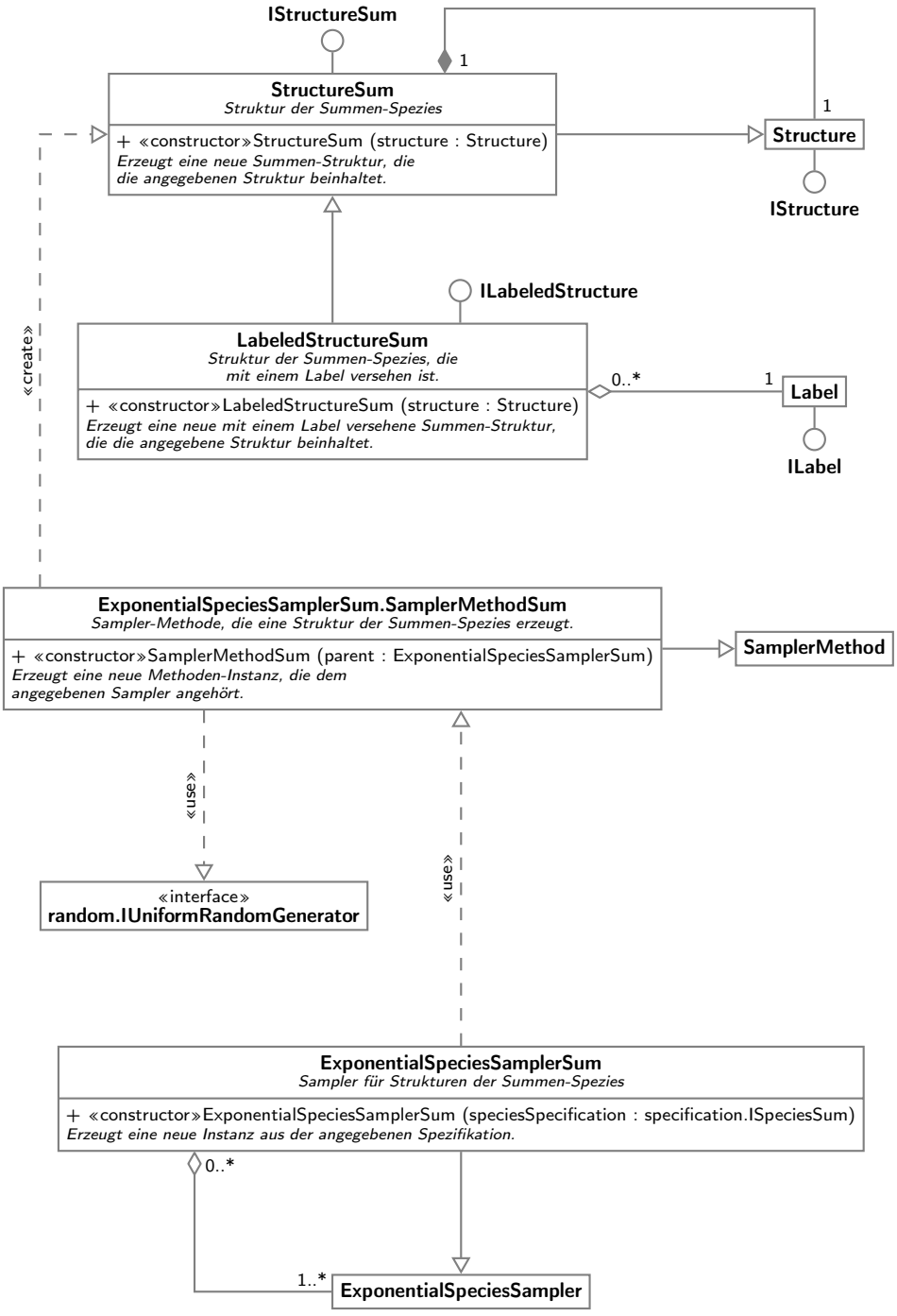

Abbildung 35: Sampler und Strukturen der Summen-Spezies

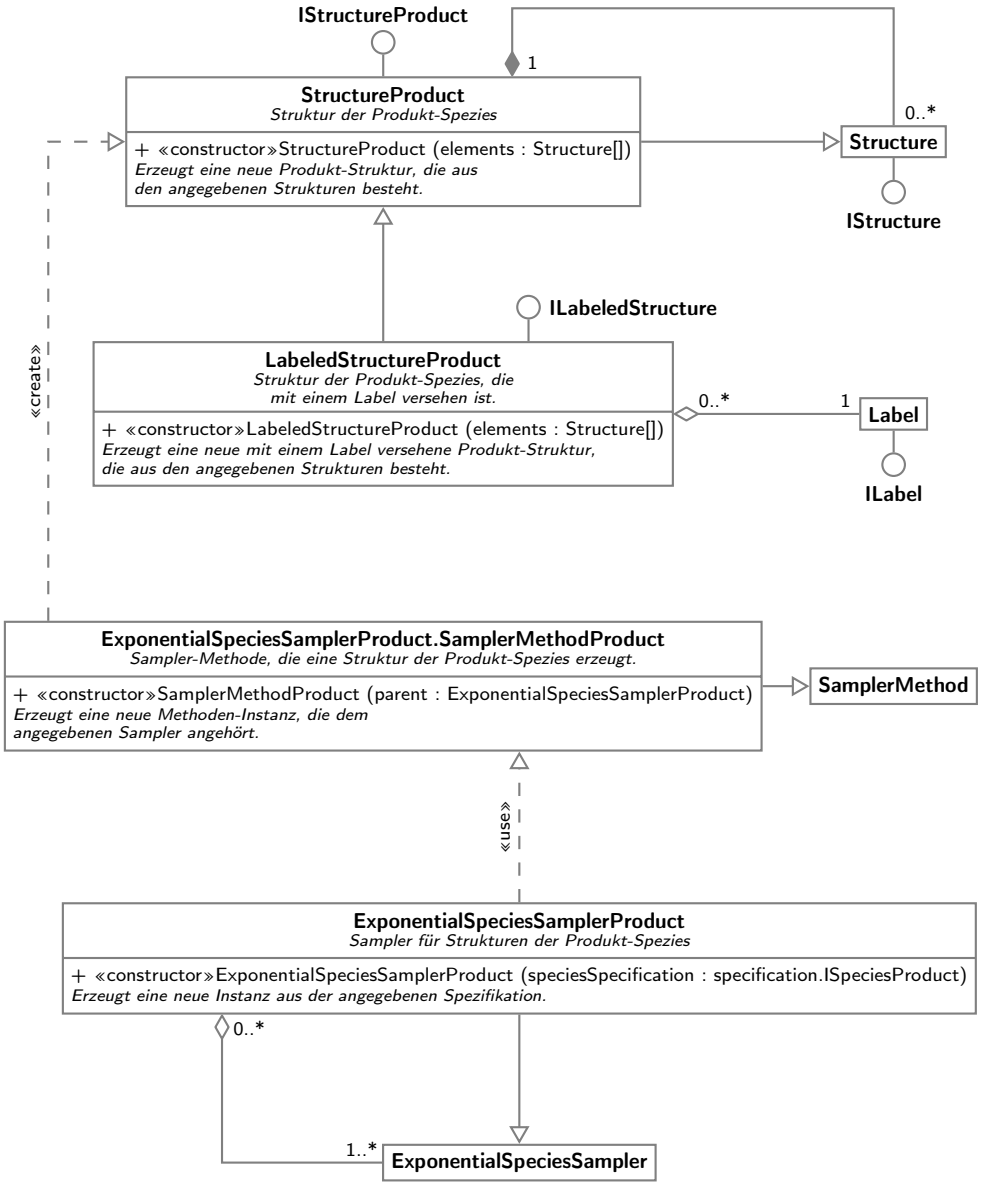

Abbildung 36: Sampler und Strukturen der Produkt-Spezies

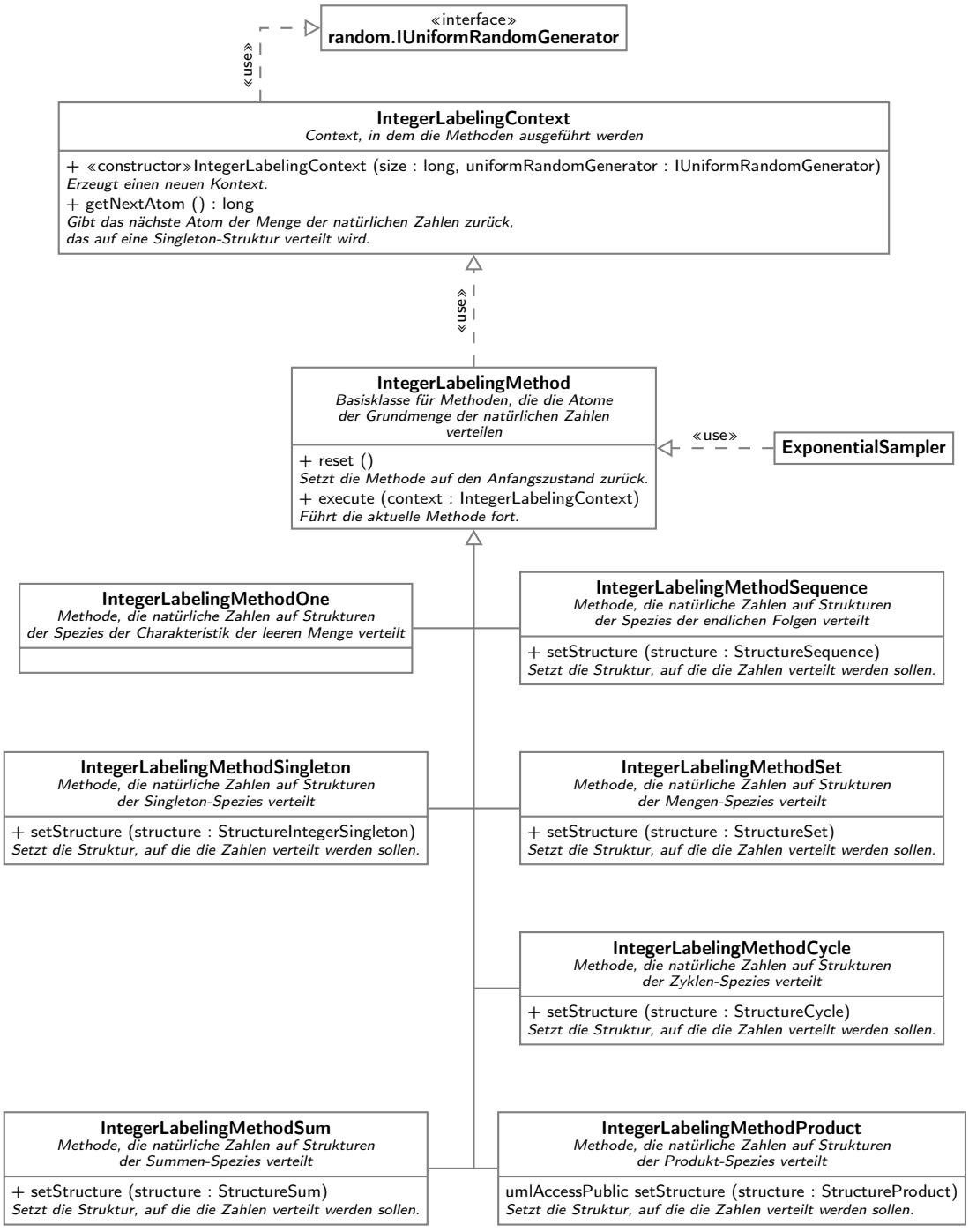

Abbildung 37: Verteilen der Zahlen der Grundmenge der natürlichen Zahlen auf die gesampelten Atome

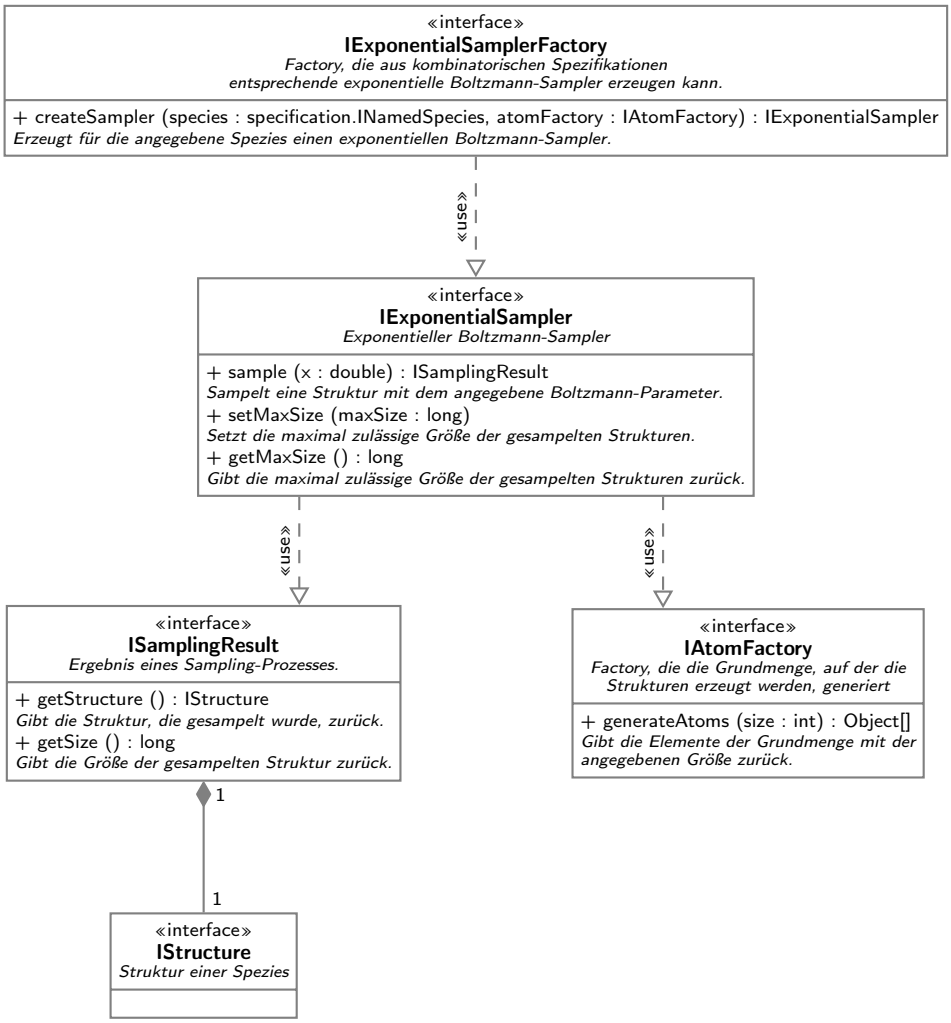

Abbildung 38: Interfaces des exponentiellen Boltzmann-Samplers

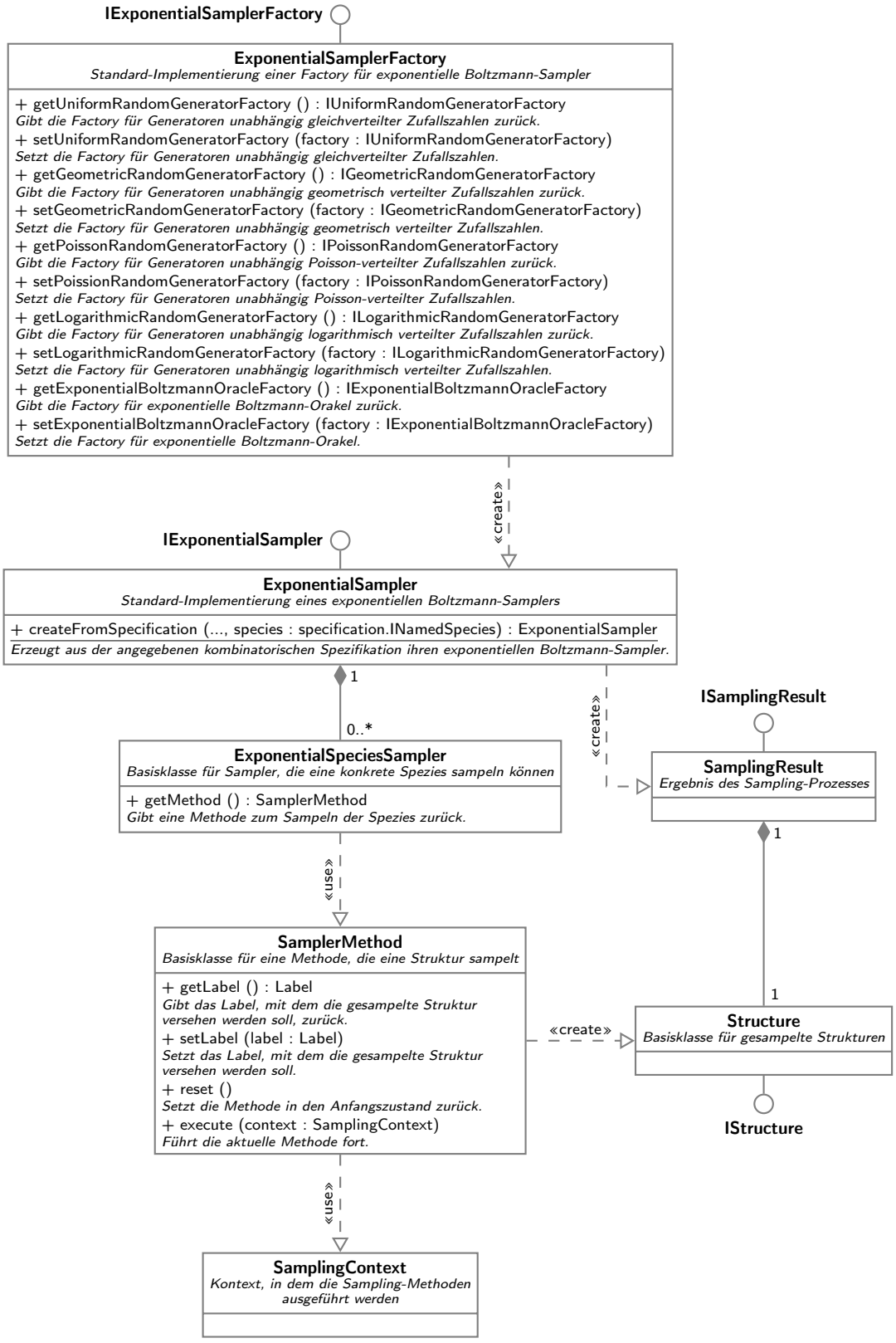

Abbildung 39: Implementierung des exponentiellen Boltzmann-Samplers

4.7 Ausgabe der gesampelten Strukturen

Das Package *at.techmath.boltzmann.output* enthält Interfaces und Klassen, die zur Ausgabe der gesampelten Strukturen dienen. In Abbildung 41 sehen wir das zentrale Interface *IStructureOutput*. Dieses stellt eine Methode *outputSamplingResult* zur Verfügung, die die übergebene Struktur im jeweils implementierten Format in den übergebenen Stream ausgibt. Es wurden zwei verschiedene Ausgaben implementiert.

Das erste Ausgabeformat wird von der Klasse *StructureOutputTotalSize* realisiert. Hierbei handelt es sich um die alleinige Ausgabe der Strukturgröße im Format `<Größe> <Zeilenumbruch>`, wobei die Größe dezimal ausgegeben wird. Diese Ausgabeklasse wird verwendet, wenn das Programm mit dem Argument `-m totalsize` aufgerufen wird.

Beispiel 4.7.1. Angenommen, der Sampler hat 5 Strukture mit den jeweiligen Größen 1253, 2387, 968, 1890 und 1345 erzeugt, so lautet die Ausgabe bei Verwendung der *StructureOutputTotalSize*-Klasse:

```
1253
2387
968
1890
1345
```

Das Hauptausgabeformat wird von der Klasse *StructureOutputText* erzeugt. Bei diesem Format handelt es sich um ein Textformat, das mittels Angabe der Algorithmen, die für die jeweiligen Struktur-Typen die Ausgabe erzeugen, beschrieben wird. Diese Klasse wird verwendet, wenn das Programm mit dem Argument `-m text` aufgerufen wird.

In Abbildung 40 sehen wir die Klassenstruktur. Der übliche Ansatz, den Baum der gesampelten Strukturen rekursiv abzuarbeiten kann aufgrund der möglichen Gröse der gesampelten Strukturen zu dem Problem führen, dass der Stack überläuft. Aus diesem Grund wurde der Algorithmus derart formuliert, dass ein eigener Stack angelegt wird, und Methoden als Klassen modelliert sind, deren Instanzen sich auf diesem Stack befinden. Nun kann die Rekursion simuliert werden, indem in einer While-Schleife die jeweils erste Methode am Stack weitergeführt wird, bis der Stack leer ist. Dieser Stack sowie alle für die Ausführung einer Ausgabemethode notwendigen Informationen werden von der *StructureOutputTextContext*-Klasse verwaltet.

Die Methoden sind alle von der *StructureOutputTextMethod*-Klasse abgeleitet. Diese stellt einen Automaten dar, dessen Zustände die einzelnen Funktionsblöcke des abzubildenden Algorithmus zwischen den Aufrufen von anderen Methoden repräsentieren. Die Methode *reset* versetzt die Methoden-Instanz hierbei in den Anfangszustand, die Methode *execute* führt die Methode im aktuellen Zustand weiter. Muss nun eine andere Methoden-Instanz aufgerufen werden, so wird diese von der aktuellen Methoden-Instanz auf den Stack des Kontexts gelegt, in den nächsten Zustand geschalten und zum Aufrufer zurückgekehrt. Wir wenden uns nun den Algorithmen zur Ausgabe der Strukturen der verschiedenen Spezies zu.

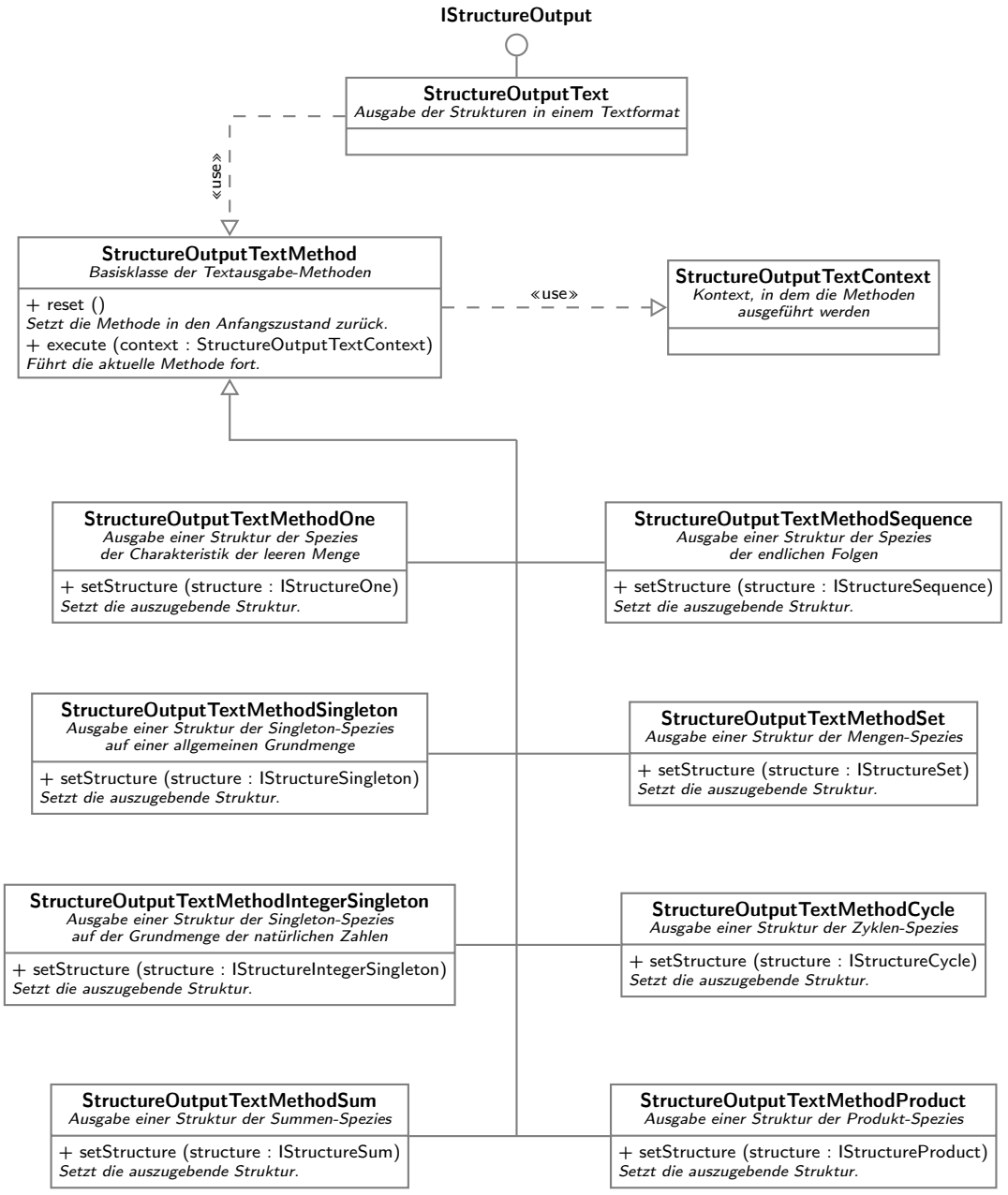

Abbildung 40: Textausgabe

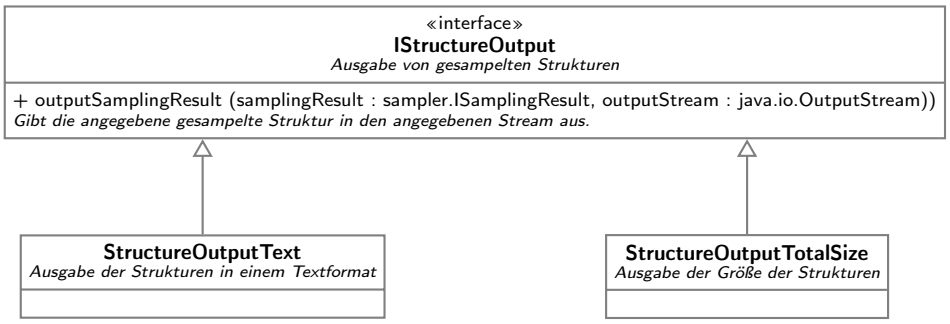

Abbildung 41: Ausgabe der Strukturen

Alle Strukturen haben gemeinsam, dass es jeweils eine mit einer Marke versehene Version und eine ohne Marke gibt. Falls eine Marke vorhanden ist, dann wird diese an die Ausgabe der eigentlichen Struktur angehängt. Die Marke wird mit Algorithmus 12 ausgegeben.

Struktur der Spezies der Charakteristik der leeren Menge Diese Struktur wird von Algorithmus 13 ausgegeben (Klasse *StructureOutputTextMethodOne*). Betrachen wir zum Beispiel eine Struktur, die mit einer ganzzahligen Marke mit dem Wert 100 versehen ist, so lautet die Ausgabe `on100`, ist sie mit einer Textmarke mit dem Inhalt `'sometext'` versehen, so lautet die Ausgabe `ot"sometext"`. Eine Struktur ohne Marke wird als `o` ausgegeben.

Struktur der Singleton-Spezies Singleton-Strukturen werden vom Algorithmus 14 ausgegeben (Klasse *StructureOutputTextMethodSingleton* beziehungsweise *StructureOutputTextMethodIntegerSingleton*). Eine Singleton-Struktur auf der Grundmenge der natürlichen Zahlen, die mit dem Atom 1234 versehen ist, und eine Textmarke `'intern'` trägt, wird beispielweise als `a1234t"intern"` ausgegeben, eine Struktur mit dem Atom 89690 und der ganzzahligen Marke 799 wird als `a89690n799` ausgegeben. Die Ausgabe einer Struktur, die keine Marke trägt und mit dem Atom 9867 versehen ist, lautet `a9876`.

Struktur der Summen-Spezies Die Ausgabe der Strukturen der Summen-Spezies erfolgt durch Algorithmus 15, der von der Klasse *StructureOutputTextMethodSum* implementiert wird. Sie besteht aus einem `'+'`-Zeichen, der Ausgabe der referenzierten Struktur und der Ausgabe der optionalen Marke. Wurde vom Sampler der Summen-Spezies beispielsweise eine Singleton-Struktur mit Atom 123456 und ganzzahliger Marke 543 erzeugt, und wurde die Summen-Struktur im weiteren Verlauf mit der Textmarke `'summe'` versehen, so lautet der erzeugte Text `+a123456n543t"summe"`.

Struktur der Produkt-Spezies Produkt-Strukturen werden vom Algorithmus 16 ausgegeben, der von der Klasse *StructureOutputTextMethodProduct* implementiert wird. Die Ausgabe besteht aus dem Zeichen `'<'`, einer Liste der Ausgaben der enthaltenen Strukturen, die durch `','` getrennt ist, dem Zeichen `'>'` und der Ausgabe der optionalen Marke, mit der die Struktur versehen sein kann. So wird zum Beispiel das mit der Marke 200 versehene Produkt aus den Singleton-Strukturen mit Atomen 123, 234 und 321, die jeweils mit der ganzzahligen Markierung 100 versehen sind, als `<a123n100,a234n100,a321n100>n200` ausgegeben.

Struktur der Spezies der endlichen Folgen Für die Ausgabe von Strukturen der Folgen-Spezies ist der durch die Klasse *StructureOutputTextMethodSequence* implementierte Algorithmus 17 verantwortlich. Die Ausgabe besteht aus dem Zeichen `'('`, einer durch `','` getrennte Liste

der Ausgaben der enthaltenen Strukturen, dem Zeichen `')'` und der Ausgabe der optionalen Marke der Struktur. Eine mit der Textmarke `'folge'` versehene Folge der Singleton-Strukturen, die mit den Atomen $1, 2, 3, 4$ und 5 versehen ist und keine Marke besitzen, führt zur Ausgabe `(a1,a2,a3,a4,a5)t"folge"`.

Struktur der Mengen-Spezies Die Klasse *StructureOutputTextMethodSet* implementiert den Algorithmus 18, der Strukturen der Mengen-Spezies ausgibt. Die Ausgabe besteht aus dem Zeichen `'{'`, einer durch `','` getrennte Liste der Ausgaben der in der Menge enthaltenen Strukturen, dem Zeichen `'}'` und der Ausgabe der optionalen Marke, mit der die Mengen-Struktur versehen ist. Eine mit der Textmarke `'menge'` versehene Menge von Singleton-Strukturen, die mit den Atomen $1, 2, 3$ und 4 versehen ist und die Textmarke `'atom'` besitzen, wird als `{a1t"atom",a2t"atom",a3t"atom",a4t"atom"}t"menge"` ausgegeben.

Struktur der Zyklen-Spezies Die letzte Strukturausgabe, die wir betrachten, ist die der Strukturen der Zyklen-Spezies. Diese wird durch Algorithmus 19 bewerkstelltigt, der von der Klasse *StructureOutputTextMethodCycle* implementiert wird. Die Ausgabe einer Zyklen-Struktur setzt sich aus dem Zeichen `'['`, einer durch `','` getrennte Liste der Ausgaben der Strukturen, aus denen sich der Zyklus zusammensetzt, dem Zeichen `']'` und der Ausgabe der optionalen Marke der Zyklen-Struktur zusammen. Als Beispiel betrachten wir die Ausgabe eines Zyklus, der aus den sieben Singleton-Strukturen mit den Atomen $1, 3, 5, 7, 2, 4$ und 6 besteht, wobei keine der Strukturen mit einer zusätzlichen Marke versehen ist. Die Ausgabe dieser Struktur lautet `[a1,a3,a5,a7,a2,a4]`.

Algorithmus 12 Ausgabe des Labels einer markierten Struktur

Eingabe: Instanz i des *ILabeledStructure*-Interfaces
Ausgabe: Ausgabe des Labels der Struktur
  $label \leftarrow i.getLabel()$
  if *label* ist eine *IIntegerLabel*-Instanz **then**
    Ausgabe von `'n'`
    Ausgabe von $label.getNumber()$ in dezimal
  else if *label* ist eine *IStringLabel*-Instanz **then**
    Ausgabe von `'t'` und `'"'`
    Ausgabe von $label.getText()$
    Ausgabe von `'"'`
  end if

Algorithmus 13 Ausgabe einer Struktur der Spezies der Charakteristik der leeren Menge

Eingabe: Instanz s des *IStructureOne*-Interfaces
Ausgabe: Ausgabe der Struktur
  Ausgabe von `'o'`
  if s ist Instanz des *ILabeledStructure*-Interfaces **then**
    Ausgabe des Labels
  end if

Algorithmus 14 Ausgabe einer Struktur der Singleton-Spezies

Eingabe: Instanz s des *IStructureSingleton*-Interfaces
Ausgabe: Ausgabe der Struktur
  Ausgabe von `'a'`
  if s ist *IStructureIntegerSingleton*-Instanz **then**
    Ausgabe von $s.getNumber()$ in dezimal
  else
    Ausgabe von $s.getAtom().toString()$
  end if
  if s ist Instanz des *ILabeledStructure*-Interfaces **then**
    Ausgabe des Labels
  end if

Algorithmus 15 Ausgabe einer Summen-Struktur

Eingabe: Instanz s des *IStructureSum*-Interfaces
Ausgabe: Ausgabe der Struktur
  Ausgabe von `'+'`
  Ausgabe der Struktur $s.getStructure$
  if s ist Instanz des *ILabeledStructure*-Interfaces **then**
    Ausgabe des Labels
  end if

Algorithmus 16 Ausgabe einer Produkt-Struktur

Eingabe: Instanz s des *IStructureProduct*-Interfaces
Ausgabe: Ausgabe der Struktur
  Ausgabe von `'<'`
  $count \leftarrow s.getCount()$
  if $count > 0$ **then**
    Ausgabe der Struktur $s.getElement(0)$
    if $count > 1$ **then**
      for $i = 1 \,..\, count - 1$ **do**
        Ausgabe von `','`
        Ausgabe der Struktur $s.getElement(i)$
      end for
    end if
  end if
  Ausgabe von `'>'`
  if s ist Instanz des *ILabeledStructure*-Interfaces **then**
    Ausgabe des Labels
  end if

Algorithmus 17 Ausgabe einer SEQ-Struktur

Eingabe: Instanz s des *IStructureSequence*-Interfaces
Ausgabe: Ausgabe der Struktur

```
Ausgabe von '('
count ← s.getCount()
if count > 0 then
    Ausgabe der Struktur s.getElement(0)
    if count > 1 then
        for i = 1 .. count − 1 do
            Ausgabe von ','
            Ausgabe der Struktur s.getElement(i)
        end for
    end if
end if
Ausgabe von ')'
if s ist Instanz des ILabeledStructure-Interfaces then
    Ausgabe des Labels
end if
```

Algorithmus 18 Ausgabe einer SET-Struktur

Eingabe: Instanz s des *IStructureSet*-Interfaces
Ausgabe: Ausgabe der Struktur

```
Ausgabe von '{'
count ← s.getCount()
if count > 0 then
    Ausgabe der Struktur s.getElement(0)
    if count > 1 then
        for i = 1 .. count − 1 do
            Ausgabe von ','
            Ausgabe der Struktur s.getElement(i)
        end for
    end if
end if
Ausgabe von '}'
if s ist Instanz des ILabeledStructure-Interfaces then
    Ausgabe des Labels
end if
```

Algorithmus 19 Ausgabe einer Cyc-Struktur

Eingabe: Instanz s des *IStructureCycle*-Interfaces
Ausgabe: Ausgabe der Struktur

```
Ausgabe von '['
count ← s.getCount()
if count > 0 then
    Ausgabe der Struktur s.getElement(0)
    if count > 1 then
        for i = 1 .. count − 1 do
            Ausgabe von ','
            Ausgabe der Struktur s.getElement(i)
        end for
    end if
end if
Ausgabe von ']'
if s ist Instanz des ILabeledStructure-Interfaces then
    Ausgabe des Labels
end if
```

A Anhang

A.1 Source-Code

A.1.1 Package at.techmath.boltzmann.app.console

Listing 2: Klasse *at.techmath.boltzmann.app.console.ConsoleMain*

```
1  package at.techmath.boltzmann.app.console;
2
3  import java.io.FileNotFoundException;
4  import java.io.IOException;
5  import java.io.OutputStream;
6  import java.io.FileOutputStream;
7
8  import at.techmath.boltzmann.specification.*;
9  import at.techmath.boltzmann.oracle.*;
10 import at.techmath.boltzmann.sampler.*;
11 import at.techmath.boltzmann.output.*;
12
13 /**
14  * Consolen-Anwendung.
15  * @author Stefan Schnabl (e0226245)
16  */
17 public class ConsoleMain {
18     /**
19      * @param args Argumente, mit denen das Programm aufgerufen wird.
20      */
21     public static void main(String[] args) {
22         // Laden der Kommandozeilen-Argumente
23         CommandLineConfiguration config = null;
24         try {
25             config = CommandLineConfiguration.createFromCommandLineArguments(args);
26             System.err.println (config.toString());
27             System.err.println();
28         } catch (Exception ex) {
29             System.err.println (ex.getMessage());
30             System.err.println ();
31             CommandLineConfiguration.printUsage(System.err);
32             return;
33         }
34
35         // Spezifikations-Kollektion laden
36         ISpecificationCollection specificationCollection = null;
37         try {
38             SpecificationCollectionXmlLoader loader = new SpecificationCollectionXmlLoader();
39             specificationCollection = loader.loadFromFile(config.getSpecificationFilename());
40         } catch (Exception ex) {
41             System.err.println (ex.getMessage());
42             return;
43         }
44
45         // Spezifikation laden
46         ISpecification specification = config.getSpecificationName() == null ?
47             specificationCollection.getDefaultSpecification() :
48             specificationCollection.getSpecification(config.getSpecificationName());
49         if (specification == null) {
50             System.err.println("Keine Spezifikation gefunden.");
51         } // if (specification == null) ...
52
53         if (config.getOutputMode() == CommandLineConfiguration.OUTPUT_MODE_ORACLE) {
54             double boltzmannParameter = config.getBoltzmannParameter();
55             System.err.println("Boltzmann-Orakel zum Parameter " + boltzmannParameter);
56             IExponentialBoltzmannOracleFactory oracleFactory = new
                   DefaultExponentialBoltzmannOracleFactory();
57             IExponentialBoltzmannOracle oracle = oracleFactory.createOracle(specification);
58             int speciesCount = specification.getCount();
59             try {
60                 for (int c = 0; c < speciesCount; c++) {
61                     INamedSpecies currentSpecies = specification.getSpeciesIndex(c);
62                     System.err.println(currentSpecies.getName() + ": " + oracle.evaluate(c,
                           boltzmannParameter));
63                 } // for (int c = 0; c < speciesCount; c++) ...
64             } catch (Exception ex) {
65                 System.err.println(ex.getMessage());
66                 return;
67             }
```

```
68          } else { // if (config.getOutputMode() != CommandLineConfiguration.OUTPUT_MODE_ORACLE)
                ...
69              // Spezies, die gesampelt werden soll, ermitteln
70              INamedSpecies species = config.getSpeciesName() == null ?
71                  specification.getDefaultSpecies() :
72                  specification.getSpecies(config.getSpeciesName());
73              if (species == null) {
74                  System.err.println("Keine Spezies gefunden.");
75              } // if (species == null) ...
76
77              FileOutputStream fileStream = null;
78              try {
79                  if (config.getOutputFile() != null) {
80                      try {
81                          fileStream = new FileOutputStream(config.getOutputFile(), true);
82                      } catch (FileNotFoundException fileNotFoundException) {
83                          System.err.println (fileNotFoundException.getMessage());
84                          fileStream = null;
85                          return;
86                      }
87                  } // if (config.getOutputFile() != null) ...
88
89                  OutputStream outputStream = fileStream == null ? System.out : fileStream;
90
91                  IExponentialSamplerFactory samplerFactory = new ExponentialSamplerFactory();
92                  IExponentialSampler sampler = samplerFactory.createSampler(species, null);
93                  sampler.setMaxSize(config.getMaxSize());
94
95                  IStructureOutput output =
96                      config.getOutputMode() == CommandLineConfiguration.OUTPUT_MODE_TEXT ?
97                          new StructureOutputText() :
98                          new StructureOutputTotalSize();
99
100                 long maxTries = config.getMaxTries();
101                 long minSize = config.getMinSize();
102                 long maxSize = config.getMaxSize();
103                 long structureCount = config.getCount();
104                 long currentStructureCount = 0L;
105                 double boltzmannParameter = config.getBoltzmannParameter();
106                 long c;
107                 for (c = 0; c < maxTries && currentStructureCount < structureCount; c++) {
108                     ISamplingResult samplingResult;
109                     try {
110                         samplingResult = sampler.sample(boltzmannParameter);
111                     } catch (at.techmath.boltzmann.sampler.StructureSizeExceededException
                            sizeException) {
112                         samplingResult = null;
113                     }
114                     if (samplingResult != null) {
115                         long resultSize = samplingResult.getSize();
116                         if (resultSize >= minSize && resultSize <= maxSize) {
117                             try {
118                                 output.outputSamplingResult(samplingResult, outputStream);
119                             } catch (IOException ioException) {
120                                 System.err.println (ioException.getMessage());
121                                 return;
122                             }
123                             currentStructureCount++;
124                         } // if (resultSize >= minSize && resultSize <= maxSize) ...
125                     } // if (samplingResult != null) ...
126                     if ((c + 1) % 1000 == 0) {
127                         System.out.println ("" + (c+1) + " Strukturen gesampelt, " +
                                currentStructureCount + " Treffer.");
128                     }
129                 } // for (long c = 0; c < maxTries && currentStructureCount < structureCount; c
                        ++) ...
130                 System.out.println ("" + c + " Strukturen gesampelt, " + currentStructureCount
                        + " Treffer.");
131             } finally {
132                 if (fileStream != null) {
133                     try { fileStream.flush(); } catch (IOException ioException) { System.err.
                            println(ioException.getMessage()); }
134                     try { fileStream.close(); } catch (IOException ioException) { System.err.
                            println(ioException.getMessage()); }
135                 } // if (fileStream != null) ...
136             }
137         } // if (config.getOutputMode() != config.OUTPUT_MODE_ORACLE) ...
138     } // public static void main(String[] args) ...
139 } // public class ConsoleMain ...
```

Listing 3: Klasse *at.techmath.boltzmann.app.console.CommandLineConfiguration*

```
1  package at.techmath.boltzmann.app.console;
2
3  import java.io.OutputStream;
4  import java.io.PrintStream;
5
6  /**
7   * Konfiguration, die aus den Kommandozeilen-Parametern geladen wird.
8   * @author Stefan Schnabl (e0226245)
9   */
10 class CommandLineConfiguration {
11     /**
12      * Kommandozeilen-Parameter fuer den Dateinamen der Spezifikation.
13      */
14     private static final String COMMAND_LINE_SWITCH_SPECIFICATION_FILENAME = "-d"; // Steht
           fuer Definition
15
16     /**
17      * Muss der Dateiname der Spezifikation angegeben werden?
18      */
19     private static final boolean REQUIRED_SPECIFICATION_FILENAME = true;
20
21     /**
22      * Standardwert des Dateinamens der Spezifikationsdatei.
23      */
24     private static final String DEFAULT_SPECIFICATION_FILENAME = null;
25
26     /**
27      * Dateiname der Spezifikationsdatei.
28      */
29     private String _specificationFilename = DEFAULT_SPECIFICATION_FILENAME;
30
31     /**
32      * Gibt den Dateinamen der Spezifikationsdatei zurueck.
33      * @return Dateiname der Spezifikationsdatei. Wenn null, dann ist kein
34      *   Spezifikationsdateiname angegeben.
35      */
36     public String getSpecificationFilename() {
37         return _specificationFilename;
38     } // public String getSpecificationFilename() ...
39
40     /**
41      * Kommandozeilen-Parameter fuer den Namen der Spezifikation.
42      */
43     private static final String COMMAND_LINE_SWITCH_SPECIFICATION_NAME = "-n"; // Steht fuer
           Name der Spezifikation
44
45     /**
46      * Muss der Name der Spezifikation angegeben werden?
47      */
48     private static final boolean REQUIRED_SPECIFICATION_NAME = false;
49
50     /**
51      * Standardwert des Namens der Spezifikation.
52      */
53     private static final String DEFAULT_SPECIFICATION_NAME = null;
54
55     /**
56      * Name der Spezifikation.
57      */
58     private String _specificationName = DEFAULT_SPECIFICATION_NAME;
59
60     /**
61      * Gibt den Namen der Spezifikation zurueck.
62      * @return Name der Spezifikation. Wenn null, dann ist keine
63      *   Spezifikation angegeben.
64      */
65     public String getSpecificationName() {
66         return _specificationName;
67     } // public String getSpecificationName() ...
68
69     /**
70      * Kommandozeilen-Parameter fuer den Namen der Spezies.
71      */
72     private static final String COMMAND_LINE_SWITCH_SPECIES_NAME = "-s"; // Steht fuer "Spezies
           "
73
74     /**
75      * Muss der Name der Spezies angegeben werden?
76      */
```

```
77      private static final boolean REQUIRED_SPECIES_NAME = false;
78
79      /**
80       * Standardwert des Namens der Spezies.
81       */
82      private static final String DEFAULT_SPECIES_NAME = null;
83
84      /**
85       * Name der Spezies.
86       */
87      private String _speciesName = DEFAULT_SPECIES_NAME;
88
89      /**
90       * Gibt den Namen der Spezies zurueck.
91       * @return Name der Spezies. Wenn null, dann ist keine
92       *   Spezies angegeben.
93       */
94      public String getSpeciesName() {
95          return _speciesName;
96      } // public String getSpeciesName() ...
97
98      /**
99       * Kommandozeilen-Parameter fuer den Boltzmann-Parameter.
100      */
101     private static final String COMMAND_LINE_SWITCH_BOLTZMANN_PARAMETER = "-x"; // Steht fuer f
            (x) ...
102
103     /**
104      * Muss der Boltzmann-Parameter angegeben werden?
105      */
106     private static final boolean REQUIRED_BOLTZMANN_PARAMETER = true;
107
108     /**
109      * Standardwert des Boltzmann-Parameters.
110      */
111     private static final double DEFAULT_BOLTZMANN_PARAMETER = 0.0D;
112
113     /**
114      * Boltzmann-Parameter.
115      */
116     private double _boltzmannParameter = DEFAULT_BOLTZMANN_PARAMETER;
117
118     /**
119      * Gibt den Boltzmann-Parameter zurueck.
120      * @return Boltzmann-Parameter.
121      */
122     public double getBoltzmannParameter() {
123         return _boltzmannParameter;
124     } // public double getBoltzmannParameter() ...
125
126     /**
127      * Kommandozeilen-Parameter fuer den Ausgabemodus.
128      */
129     private static final String COMMAND_LINE_SWITCH_OUTPUT_MODE = "-m"; // Steht fuer Modus.
130
131     /**
132      * Muss der Ausgabemodus angegeben werden?
133      */
134     private static final boolean REQUIRED_OUTPUT_MODE = false;
135
136     /**
137      * Ausgabemodus Gesamtgroesse.
138      */
139     public static final int OUTPUT_MODE_TOTAL_SIZE = 1;
140
141     /**
142      * Parameter fuer den Ausgabemodus Gesamtgroesse.
143      */
144     private static final String OUTPUT_MODE_PARAMETER_TOTAL_SIZE = "totalsize";
145
146     /**
147      * Ausgabemodus Spezies als Text.
148      */
149     public static final int OUTPUT_MODE_TEXT = 2;
150
151     /**
152      * Parameter fuer den Ausgabemodus Text.
153      */
154     private static String OUTPUT_MODE_PARAMETER_TEXT = "text";
155
```

```
156     /**
157      * Ausgabemodus Boltzmann-Orakel.
158      */
159     public static final int OUTPUT_MODE_ORACLE = 3;
160 
161     /**
162      * Parameter fuer den Ausgabemodus Boltzmann-Orakel.
163      */
164     private static final String OUTPUT_MODE_PARAMETER_ORACLE = "oracle";
165 
166     /**
167      * Standardwert des Ausgabemodus.
168      */
169     public static final int DEFAULT_OUTPUT_MODE = OUTPUT_MODE_TEXT;
170 
171     /**
172      * Ausgabemodus.
173      */
174     private int _outputMode = DEFAULT_OUTPUT_MODE;
175 
176     /**
177      * Gibt den Ausgabemodus zurueck.
178      * @return Ausgabemodus.
179      */
180     public int getOutputMode() {
181         return _outputMode;
182     } // public int getOutputMode() ...
183 
184     /**
185      * Kommandozeilen-Parameter fuer den Namen der Ausgabedatei.
186      */
187     private static final String COMMAND_LINE_SWITCH_OUTPUT_FILE = "-f"; // Steht fuer File
188 
189     /**
190      * Muss der Name der Ausgabedatei angegeben werden?
191      */
192     private static final boolean REQUIRED_OUTPUT_FILE = false;
193 
194     /**
195      * Standardwert fuer den Namen der Ausgabedatei.
196      */
197     private static final String DEFAULT_OUTPUT_FILE = null;
198 
199     /**
200      * Name der Ausgabedatei.
201      */
202     private String _outputFile = DEFAULT_OUTPUT_FILE;
203 
204     /**
205      * Gibt den Namen der Ausgabedatei zurueck.
206      * @return Name der Ausgabedatei. Wenn null, dann ist keine
207      *   Ausgabedatei angegeben, und die Ausgabe erfolgt in die Standardausgabe.
208      */
209     public String getOutputFile() {
210         return _outputFile;
211     } // public String getOutputFile() ...
212 
213     /**
214      * Kommandozeilen-Parameter fuer die minimale Groesse der zu sampelnden Strukturen.
215      */
216     private static final String COMMAND_LINE_SWITCH_MIN_SIZE = "-min";
217 
218     /**
219      * Muss die minimale Groesse der zu sampelnden Strukturen angegeben werden?
220      */
221     private static final boolean REQUIRED_MIN_SIZE = false;
222 
223     /**
224      * Standardwert fuer die minimale Groesse der zu sampelnden Strukturen.
225      */
226     private static final long DEFAULT_MIN_SIZE = 0L;
227 
228     /**
229      * Minimale Groesse der zu sampelnden Strukturen.
230      */
231     private long _minSize = DEFAULT_MIN_SIZE;
232 
233     /**
234      * Gibt die minimale Groesse der zu sampelnden Strukturen zurueck.
235      * @return Minimale Groesse der zu sampelnden Strukturen.
```

```
236      */
237     public long getMinSize() {
238         return _minSize;
239     } // public long getMinSize() ...
240 
241     /**
242      * Kommandozeilen-Parameter fuer die maximale Groesse der zu sampelnden Strukturen.
243      */
244     private static final String COMMAND_LINE_SWITCH_MAX_SIZE = "-max";
245 
246     /**
247      * Muss die maximale Groesse der zu sampelnden Strukturen angegeben werden.
248      */
249     private static final boolean REQUIRED_MAX_SIZE = false;
250 
251     /**
252      * Standardwert fuer die maximale Groesse der zu sampelnden Strukturen.
253      */
254     private static final long DEFAULT_MAX_SIZE = Long.MAX_VALUE;
255 
256     /**
257      * Maximale Groesse der zu sampelnden Strukturen.
258      */
259     private long _maxSize = DEFAULT_MAX_SIZE;
260 
261     /**
262      * Gibt die maximale Groesse der zu sampelnden Strukturen zurueck.
263      * @return Maximale Groesse der zu sampelnden Strukturen.
264      */
265     public long getMaxSize() {
266         return _maxSize;
267     } // public long getMaxSize() ...
268 
269     /**
270      * Kommandozeilen-Parameter fuer die Anzahl der zu sampelnden Strukturen.
271      */
272     private static final String COMMAND_LINE_SWITCH_COUNT = "-c";
273 
274     /**
275      * Muss die Anzahl der zu sampelnden Strukturen angegeben werden?
276      */
277     private static final boolean REQUIRED_COUNT = false;
278 
279     /**
280      * Standardwert fuer die Anzahl der zu sampelnden Strukturen.
281      */
282     private static final long DEFAULT_COUNT = 1L;
283 
284     /**
285      * Anzahl der zu sampelnden Strukturen.
286      */
287     private long _count = DEFAULT_COUNT;
288 
289     /**
290      * Gibt die Anzahl der zu sampelnden Strukturen zurueck.
291      * @return Anzahl der zu sampelnden Strukturen.
292      */
293     public long getCount() {
294         return _count;
295     } // public long getCount() ...
296 
297     /**
298      * Kommandozeilen-Parameter fuer die maximale Anzahl der Versuche.
299      */
300     private static final String COMMAND_LINE_SWITCH_MAX_TRIES = "-t"; // Steht fuer Tries.
301 
302     /**
303      * Muss die maximale Anzahl der Versuche angegeben werden?
304      */
305     private static final boolean REQUIRED_MAX_TRIES = false;
306 
307     /**
308      * Standardwert fuer die maximale Anzahl der Versuche.
309      */
310     private static long DEFAULT_MAX_TRIES = Long.MAX_VALUE;
311 
312     /**
313      * Maximale Anzahl der Versuche.
314      */
315     private long _maxTries = DEFAULT_MAX_TRIES;
```

```
316
317     /**
318      * Gibt die maximale Anzahl der Versuche zurueck.
319      * @return Maximale Anzahl der Versuche.
320      */
321     public long getMaxTries() {
322         return _maxTries;
323     } // public long getMaxTries() ...
324
325     /**
326      * Erzeugt aus den uebergebenen Kommandozeilen-Argumenten eine neue Instanz.
327      * @param args Kommandozeilen-Argumente.
328      * @return Neue Instanz, die aus den uebergebenen Kommandozeilen-Argumenten erzeugt wurde.
329      * @throws IllegalArgumentException args ist null oder die Parameter fehlerhaft.
330      */
331     @SuppressWarnings("unused")
332     public static CommandLineConfiguration createFromCommandLineArguments(String[] args) {
333         if (args == null) {
334             throw new IllegalArgumentException("args = null.");
335         } // if (args == null) ...
336         // Ergebnis-Instanz erzeugen
337         CommandLineConfiguration result = new CommandLineConfiguration();
338
339         // Wurden die Parameter geladen?
340         boolean loadedSpecificationFilename = false; // Wurde der Dateiname der Spezifikation
                geladen?
341         boolean loadedSpecificationName = false; // Wurde der Name der zu sampelnden
                Spezifikation geladen?
342         boolean loadedSpeciesName = false; // Wurde der Name der Spezies geladen?
343         boolean loadedBoltzmannParameter = false; // Wurde der Boltzmann-Parameter geladen?
344         boolean loadedOutputMode = false; // Wurde der Ausgabemodus geladen?
345         boolean loadedOutputFile = false; // Wurde der Name der Ausgabedatei geladen?
346         boolean loadedMinSize = false; // Wurde die minimale Groesse der zu sampelnden
                Strukturen geladen?
347         boolean loadedMaxSize = false; // Wurde die maximale Groesse der zu sampelnden
                Strukturen geladen?
348         boolean loadedCount = false; // Wurde die Anzahl der zu sampelnden Strukturen geladen?
349         boolean loadedMaxTries = false; // Wurde die maximale Anzahl der Versuche geladen?
350
351         int argCount = args.length;
352         int currentIndex = 0;
353         while (currentIndex < argCount) {
354             String currentArgument = args[currentIndex];
355             if (currentArgument == null) { currentArgument = ""; } else { currentArgument =
                    currentArgument.trim(); }
356             if (currentArgument.equals(COMMAND_LINE_SWITCH_SPECIFICATION_FILENAME)) {
357                 // Laden des Dateinamens der Spezifikation.
358                 if (loadedSpecificationFilename) {
359                     throw new IllegalArgumentException("Dateiname der Spezifikation mehrfach
                            angegeben.");
360                 } // if (loadedSpecificationFilename) ...
361                 String filename = null;
362                 currentIndex++;
363                 if (currentIndex < argCount) {
364                     filename = args[currentIndex];
365                     if (filename != null) {
366                         filename = filename.trim();
367                         if (filename.length() <= 0) {
368                             filename = null;
369                         } // if (filename.length() <= 0) ...
370                     } // if (filename != null) ...
371                 } // if (currentIndex < argCount) ...
372                 if (filename == null) {
373                     throw new IllegalArgumentException("Ungueltiger Dateiname der Spezifikation
                            .");
374                 } // if (filename == null) ...
375                 result._specificationFilename = filename;
376                 loadedSpecificationFilename = true;
377                 currentIndex++;
378             } else if (currentArgument.equals(COMMAND_LINE_SWITCH_SPECIFICATION_NAME)) {
379                 // Laden der zu sampelnden Spezifikation.
380                 if (loadedSpecificationName) {
381                     throw new IllegalArgumentException("Name der zu sampelnden Spezifikation
                            mehrfach angegeben.");
382                 } // if (loadedSpecificationName) ...
383                 String specificationName = null;
384                 currentIndex++;
385                 if (currentIndex < argCount) {
386                     specificationName = args[currentIndex];
387                     if (specificationName != null) {
```

```
388                         specificationName = specificationName.trim();
389                         if (specificationName.length() <= 0) {
390                             specificationName = null;
391                         } // if (specificationName.length() <= 0) ...
392                     } // if (specificationName != null) ...
393                 } // if (currentIndex < argCount) ...
394                 if (specificationName == null) {
395                     throw new IllegalArgumentException("Ungueltiger Name der zu sampelnden
                         Spezifikation.");
396                 } // if (specificationName == null) ...
397                 result._specificationName = specificationName;
398                 loadedSpecificationName = true;
399                 currentIndex++;
400             } else if (currentArgument.equals(COMMAND_LINE_SWITCH_SPECIES_NAME)) {
401                 // Laden des Namens der zu sampelnden Spezies.
402                 if (loadedSpeciesName) {
403                     throw new IllegalArgumentException("Name der zu sampelnden Spezies mehrfach
                         angegeben.");
404                 } // if (loadedSpeciesName) ...
405                 String speciesName = null;
406                 currentIndex++;
407                 if (currentIndex < argCount) {
408                     speciesName = args[currentIndex];
409                     if (speciesName != null) {
410                         speciesName = speciesName.trim();
411                         if (speciesName.length() <= 0) {
412                             speciesName = null;
413                         } // if (speciesName.length() <= 0) ...
414                     } // if (speciesName != null) ...
415                 } // if (currentIndex < argCount) ...
416                 if (speciesName == null) {
417                     throw new IllegalArgumentException("Ungueltiger Name der zu sampelnden
                         Spezies.");
418                 } // if (speciesName == null) ...
419                 result._speciesName = speciesName;
420                 loadedSpeciesName = true;
421                 currentIndex++;
422             } else if (currentArgument.equals(COMMAND_LINE_SWITCH_BOLTZMANN_PARAMETER)) {
423                 // Laden des Boltzmann-Parameters.
424                 if (loadedBoltzmannParameter) {
425                     throw new IllegalArgumentException("Boltzmann-Parameter mehrfach angegeben.
                         ");
426                 } // if (loadedBoltzmannParameter) ...
427                 double boltzmannParameter;
428                 currentIndex++;
429                 if (currentIndex < argCount) {
430                     String paramString = args[currentIndex];
431                     if (paramString == null) { paramString = ""; } else { paramString =
                         paramString.trim(); }
432                     try {
433                         boltzmannParameter = Double.parseDouble(paramString);
434                     } catch (NumberFormatException numberFormatException) {
435                         throw new IllegalArgumentException("Boltzmann-Parameter ungueltig.");
436                     }
437                 } else { // if (currentIndex >= argCount) ...
438                     throw new IllegalArgumentException("Boltzmann-Parameter nicht angegeben.");
439                 } // if (currentIndex >= argCount) ...
440                 if (boltzmannParameter <= 0.0D) {
441                     throw new IllegalArgumentException("Boltzmann-Parameter kleiner oder gleich
                         0.");
442                 } // if (boltzmannParameter <= 0.0D) ...
443                 result._boltzmannParameter = boltzmannParameter;
444                 loadedBoltzmannParameter = true;
445                 currentIndex++;
446             } else if (currentArgument.equals(COMMAND_LINE_SWITCH_OUTPUT_MODE)) {
447                 // Laden des Ausgabemodus.
448                 if (loadedOutputMode) {
449                     throw new IllegalArgumentException("Ausgabemodus mehrfach angegeben.");
450                 } // if (loadedOutputMode) ...
451                 currentIndex++;
452                 int outputMode = -1;
453                 if (currentIndex < argCount) {
454                     String outputModeString = args[currentIndex];
455                     if (outputModeString == null) { outputModeString = ""; } else {
                         outputModeString = outputModeString.trim(); }
456                     if (OUTPUT_MODE_PARAMETER_TOTAL_SIZE.equals(outputModeString)) {
457                         outputMode = OUTPUT_MODE_TOTAL_SIZE;
458                     } else if (OUTPUT_MODE_PARAMETER_TEXT.equals(outputModeString)) {
459                         outputMode = OUTPUT_MODE_TEXT;
460                     } else if (OUTPUT_MODE_PARAMETER_ORACLE.equals(outputModeString)) {
```

```
461                         outputMode = OUTPUT_MODE_ORACLE;
462                     }
463                 } // if (currentIndex < argCount) ...
464                 if (outputMode == -1) {
465                     throw new IllegalArgumentException("Ausgabemodus ungueltig.");
466                 } // if (outputMode == -1) ...
467                 result._outputMode = outputMode;
468                 loadedOutputMode = true;
469                 currentIndex++;
470             } else if (currentArgument.equals(COMMAND_LINE_SWITCH_OUTPUT_FILE)) {
471                 // Laden des Namens der Ausgabedatei.
472                 if (loadedOutputFile) {
473                     throw new IllegalArgumentException("Name der Ausgabedatei mehrfach
                            angegeben.");
474                 } // if (loadedOutputFile) ...
475                 String outputFileName = null;
476                 currentIndex++;
477                 if (currentIndex < argCount) {
478                     outputFileName = args[currentIndex];
479                     if (outputFileName != null) {
480                         outputFileName = outputFileName.trim();
481                         if (outputFileName.length() <= 0) {
482                             outputFileName = null;
483                         } // if (outputFileName.length() <= 0) ...
484                     } // if (outputFileName != null) ...
485                 } // if (currentIndex < argCount) ...
486                 if (outputFileName == null) {
487                     throw new IllegalArgumentException("Ungueltiger Dateiname der Ausgabedatei.
                            ");
488                 } // if (outputFileName == null) ...
489                 result._outputFile = outputFileName;
490                 loadedOutputFile = true;
491                 currentIndex++;
492             } else if (currentArgument.equals(COMMAND_LINE_SWITCH_MIN_SIZE)) {
493                 // Laden der minimalen Groesse der zu sampelnden Strukturen.
494                 if (loadedMinSize) {
495                     throw new IllegalArgumentException("Minimale Groesse der zu sampelnden
                            Strukturen mehrfach angegeben.");
496                 } // if (loadedMinSize) ...
497                 long minSize;
498                 currentIndex++;
499                 if (currentIndex < argCount) {
500                     String minSizeString = args[currentIndex];
501                     if (minSizeString == null) { minSizeString = ""; } else { minSizeString =
                            minSizeString.trim(); }
502                     try {
503                         minSize = Long.parseLong(minSizeString, 10);
504                     } catch (NumberFormatException numberFormatException) {
505                         throw new IllegalArgumentException("Minimale Groesse der zu sampelnden
                                Strukturen ungueltig.");
506                     }
507                 } else { // if (currentIndex >= argCount) ...
508                     throw new IllegalArgumentException("Minimale Groesse der zu sampelnden
                            Strukturen nicht angegeben.");
509                 } // if (currentIndex >= argCount) ...
510                 if (minSize < 0L) {
511                     throw new IllegalArgumentException("Minimale Groesse der zu sampelnden
                            Strukturen kleiner 0.");
512                 } // if (minSize < 0L) ...
513                 result._minSize = minSize;
514                 loadedMinSize = true;
515                 currentIndex++;
516             } else if (currentArgument.equals(COMMAND_LINE_SWITCH_MAX_SIZE)) {
517                 // Laden der maximalen Groesse der zu sampelnden Strukturen.
518                 if (loadedMaxSize) {
519                     throw new IllegalArgumentException("Maximale Groesse der zu sampelnden
                            Strukturen mehrfach angegeben.");
520                 } // if (loadedMaxSize) ...
521                 long maxSize;
522                 currentIndex++;
523                 if (currentIndex < argCount) {
524                     String maxSizeString = args[currentIndex];
525                     if (maxSizeString == null) { maxSizeString = ""; } else { maxSizeString =
                            maxSizeString.trim(); }
526                     try {
527                         maxSize = Long.parseLong(maxSizeString, 10);
528                     } catch (NumberFormatException numberFormatException) {
529                         throw new IllegalArgumentException("Maximale Groesse der zu sampelnden
                                Strukturen ungueltig.");
530                     }
```

```
531                 } else { // if (currentIndex >= argCount) ...
532                     throw new IllegalArgumentException("Maximale Groesse der zu sampelnden
                            Strukturen nicht angegeben.");
533                 } // if (currentIndex >= argCount) ...
534                 if (maxSize < 0L) {
535                     throw new IllegalArgumentException("Maximale Groesse der zu sampelnden
                            Strukturen kleiner 0.");
536                 } // if (maxSize < 0L) ...
537                 result._maxSize = maxSize;
538                 loadedMaxSize = true;
539                 currentIndex++;
540             } else if (currentArgument.equals(COMMAND_LINE_SWITCH_COUNT)) {
541                 // Laden der Anzahl der zu sampelnden Strukturen.
542                 if (loadedCount) {
543                     throw new IllegalArgumentException("Anzahl der zu sampelnden Strukturen
                            mehrfach angegeben.");
544                 } // if (loadedCount) ...
545                 long count;
546                 currentIndex++;
547                 if (currentIndex < argCount) {
548                     String countString = args[currentIndex];
549                     if (countString == null) { countString = ""; } else { countString =
                            countString.trim(); }
550                     try {
551                         count = Long.parseLong(countString, 10);
552                     } catch (NumberFormatException numberFormatException) {
553                         throw new IllegalArgumentException("Anzahl der zu sampelnden Strukturen
                                ungueltig.");
554                     }
555                 } else { // if (currentIndex >= argCount) ...
556                     throw new IllegalArgumentException("Anzahl der zu sampelnden Strukturen
                            nicht angegeben.");
557                 } // if (currentIndex >= argCount) ...
558                 if (count < 0L) {
559                     throw new IllegalArgumentException("Anzahl der zu sampelnden Strukturen
                            kleiner 0.");
560                 } // if (count < 0L) ...
561                 result._count = count;
562                 loadedCount = true;
563                 currentIndex++;
564             } else if (currentArgument.equals(COMMAND_LINE_SWITCH_MAX_TRIES)) {
565                 // Laden der maximalen Anzahl der Versuche.
566                 if (loadedMaxTries) {
567                     throw new IllegalArgumentException("Maximale Anzahl der Versuche mehrfach
                            angegeben.");
568                 } // if (loadedMaxTries) ...
569                 long maxTries;
570                 currentIndex++;
571                 if (currentIndex < argCount) {
572                     String maxTriesString = args[currentIndex];
573                     if (maxTriesString == null) { maxTriesString = ""; } else { maxTriesString
                            = maxTriesString.trim(); }
574                     try {
575                         maxTries = Long.parseLong(maxTriesString, 10);
576                     } catch (NumberFormatException numberFormatException) {
577                         throw new IllegalArgumentException("Maximale Anzahl der Versuche
                                ungueltig.");
578                     }
579                 } else { // if (currentIndex >= argCount) ...
580                     throw new IllegalArgumentException("Maximale Anzahl der Versuche nicht
                            angegeben.");
581                 } // if (currentIndex >= argCount) ...
582                 if (maxTries < 0L) {
583                     throw new IllegalArgumentException("Maximale Anzahl der Versuche kleiner 0.
                            ");
584                 } // if (maxTries < 0L) ...
585                 result._maxTries = maxTries;
586                 loadedMaxTries = true;
587                 currentIndex++;
588             } else {
589                 throw new IllegalArgumentException("Ungueltiger Parameter \"" + currentArgument
                        + "\"");
590             }
591         } // while (currentIndex < argCount) ...
592 
593         if (REQUIRED_SPECIFICATION_FILENAME && !loadedSpecificationFilename) {
594             throw new IllegalArgumentException("Dateiname der Spezifikation nicht angegeben.");
595         } // if (REQUIRED_SPECIFICATION_FILENAME && !loadedSpecificationFilename) ...
596         if (REQUIRED_SPECIFICATION_NAME && !loadedSpecificationName) {
```

```
597             throw new IllegalArgumentException("Name der zu sampelnden Spezifikation nicht
                    angegeben.");
598         } // if (REQUIRED_SPECIFICATION_NAME && !loadedSpecificationName) ...
599         if (REQUIRED_SPECIES_NAME && !loadedSpeciesName) {
600             throw new IllegalArgumentException("Name der zu sampelnden Spezies nicht angegeben.
                    ");
601         } // if (REQUIRED_SPECIES_NAME && !loadedSpeciesName) ...
602         if (REQUIRED_BOLTZMANN_PARAMETER && !loadedBoltzmannParameter) {
603             throw new IllegalArgumentException("Boltzmann-Parameter nicht angegeben.");
604         } // if (REQUIRED_BOLTZMANN_PARAMETER && !loadedBoltzmannParameter) ...
605         if (REQUIRED_OUTPUT_MODE && !loadedOutputMode) {
606             throw new IllegalArgumentException("Ausgabemodus nicht angegeben.");
607         } // if (REQUIRED_OUTPUT_MODE && !loadedOutputMode) ...
608         if (REQUIRED_OUTPUT_FILE && !loadedOutputFile) {
609             throw new IllegalArgumentException("Name der Ausgabedatei nicht angegeben.");
610         } // if (REQUIRED_OUTPUT_FILE && !loadedOutputFile) ...
611         if (REQUIRED_MIN_SIZE && !loadedMinSize) {
612             throw new IllegalArgumentException("Minimale Groesse der zu sampelnden Strukturen
                    nicht angegeben.");
613         } // if (REQUIRED_MIN_SIZE && !loadedMinSize) ...
614         if (REQUIRED_MAX_SIZE && !loadedMaxSize) {
615             throw new IllegalArgumentException("Maximale Groesse der zu sampelnden Strukturen
                    nicht angegeben.");
616         } // if (REQUIRED_MAX_SIZE && !loadedMaxSize) ...
617         if (REQUIRED_COUNT && !loadedCount) {
618             throw new IllegalArgumentException("Anzahl der zu sampelnden Strukturen nicht
                    angegeben.");
619         } // if (REQUIRED_COUNT && !loadedCount) ...
620         if (REQUIRED_MAX_TRIES && !loadedMaxTries) {
621             throw new IllegalArgumentException("Maximale Anzahl der Versuche nicht angegeben.")
                    ;
622         } // if (REQUIRED_MAX_TRIES && !loadedMaxTries) ...
623         return result;
624     } // public static CommandLineConfiguration createFromCommandLineArguments(String[] args)
          ...
625 
626     /**
627      * Gibt eine String-Repraesentation der Instanz zurueck.
628      * @return String-Repraesentation der Instanz.
629      */
630     @Override
631     public String toString() {
632         StringBuilder sb = new StringBuilder();
633         sb.append("Kommandozeilen-Argumente:\n");
634         sb.append("SpecificationFileName: ").append(getSpecificationFilename()).append("\n");
635         sb.append("SpecificationName: ").append(getSpecificationName()).append("\n");
636         sb.append("SpeciesName: ").append(getSpeciesName()).append("\n");
637         sb.append("BoltzmannParameter: ").append(getBoltzmannParameter()).append("\n");
638         sb.append("OutputMode: ");
639         switch (getOutputMode()) {
640             case OUTPUT_MODE_TOTAL_SIZE: sb.append("TotalSize"); break;
641             case OUTPUT_MODE_TEXT: sb.append("Text"); break;
642             case OUTPUT_MODE_ORACLE: sb.append("Oracle"); break;
643             default: sb.append("Unbekannt"); break;
644         } // switch (getOutputMode()) ...
645         sb.append("\n");
646         sb.append("OutputFile: ").append(getOutputFile()).append("\n");
647         sb.append("MinSize: ").append(getMinSize()).append("\n");
648         sb.append("MaxSize: ").append(getMaxSize()).append("\n");
649         sb.append("Count: ").append(getCount()).append("\n");
650         sb.append("MaxTries: ").append(getMaxTries()).append("\n");
651         return sb.toString();
652     } // public String toString() ...
653 
654     /**
655      * Ausgabe der Verwendungs-Informationen.
656      * @param outputStream Stream, in den die Information ausgegeben werden soll.
657      * @throws IllegalArgumentException outputStream ist null.
658      */
659     public static void printUsage(OutputStream outputStream) {
660         if (outputStream == null) {
661             throw new IllegalArgumentException("outputStream = null.");
662         } // if (outputStream == null) ...
663         PrintStream printStream = null;
664         try {
665             printStream = new PrintStream(outputStream, false);
666             printStream.println("boltzmann - Boltzmann-Sampler");
667             printStream.println();
668             printStream.println("Parameter:");
669             printStream.print(COMMAND_LINE_SWITCH_SPECIFICATION_FILENAME);
```

```
670             printStream.println(" <Dateiname> ... Dateiname der Spezifikations-Datei.");
671             printStream.print(COMMAND_LINE_SWITCH_SPECIFICATION_NAME);
672             printStream.println(" <Name> ... Name der Spezifikation. Wenn nicht angegeben, wird
                      die Standard-Spezifikation der verwendet.");
673             printStream.print(COMMAND_LINE_SWITCH_SPECIES_NAME);
674             printStream.println(" <Name> ... Name der Spezies, die gesampelt werden soll. Wenn
                    nicht angegeben, wird die Standard-Spezies der Spezifikation verwendet.");
675             printStream.print(COMMAND_LINE_SWITCH_BOLTZMANN_PARAMETER);
676             printStream.println(" <Gleitkommazahl> ... Boltzmann-Paramter, mit dem gesampelt
                    werden soll.");
677             printStream.print(COMMAND_LINE_SWITCH_OUTPUT_MODE);
678             printStream.println(" <Modus> ... Ausgabemodus, wobei folgende Werte angegeben
                    werden koennen:");
679             printStream.print("  ");
680             printStream.print(OUTPUT_MODE_PARAMETER_TEXT);
681             printStream.println(" ... Die Strukturen werden als Text ausgegeben (Standard).");
682             printStream.print("  ");
683             printStream.print(OUTPUT_MODE_PARAMETER_TOTAL_SIZE);
684             printStream.println(" ... Gibt die Gesamtgroesse pro Struktur aus.");
685             printStream.print("  ");
686             printStream.print(OUTPUT_MODE_PARAMETER_ORACLE);
687             printStream.println(" ... Gibt die Werte der erzeugenden Funktionen aus.");
688             printStream.print(COMMAND_LINE_SWITCH_OUTPUT_FILE);
689             printStream.println(" <Name> ... Name der Ausgabedatei. Wenn nicht angegeben, wird
                    auf die Standardausgabe ausgegeben.");
690             printStream.print(COMMAND_LINE_SWITCH_MIN_SIZE);
691             printStream.println(" <Zahl> ... Minimale Groesse der gesampelten Strukturen.
                    Standardwert 0.");
692             printStream.print(COMMAND_LINE_SWITCH_MAX_SIZE);
693             printStream.println(" <Zahl> ... Maximale Groesse der gesampelten Strukturen.
                    Standardwert unbegrenzt.");
694             printStream.print(COMMAND_LINE_SWITCH_COUNT);
695             printStream.println(" <Zahl> ... Anzahl der Strukturen, die ausgegeben werden
                    sollen. Standardwert 1");
696             printStream.print(COMMAND_LINE_SWITCH_MAX_TRIES);
697             printStream.println(" <Zahl> ... Maximale Anzahl der Versuche, die durchgefuehrt
                    werden sollen. Standardwert unbegrenzt.");
698         } finally {
699             if (printStream != null) {
700                 printStream.flush();
701             } // if (printStream != null) ...
702         }
703     } // public static void printUsage(OutputStream outputStream) ...
704 } // class CommandLineConfiguration ...
```

A.1.2 Package at.techmath.boltzmann.linearalgebra

Listing 4: Klasse *at.techmath.boltzmann.linearalgebra.DefaultAlgorithms*

```
1  package at.techmath.boltzmann.linearalgebra;
2
3  /**
4   * Klasse, mit der ueber statische Eigenschaften, die Standard-Algorithmen
5   * definiert werden koennen.
6   * @author Stefan Schnabl (e0226245)
7   */
8  public class DefaultAlgorithms {
9      /**
10     * Feld, das den Standard-Algorithmus fuer die QR-Zerlegung speichert.
11     */
12    private static IRealQRFactorizationAlgorithm _defaultRealQRFactorizationAlgorithm = new
          DefaultRealQRFactorizationAlgorithm();
13
14    /**
15     * Gibt den Standard-Algorithmus fuer die QR-Zerlegung zurueck.
16     * @return Standard-Algorithmus fuer die QR-Zerlegung.
17     */
18    public static IRealQRFactorizationAlgorithm getDefaultRealQRFactorizationAlgorithm() {
19        return _defaultRealQRFactorizationAlgorithm;
20    } // public static getDefaultRealQRFactorizationAlgorithm() ...
21
22    /**
23     * Setzt den Standard-Algorithmus fuer die QR-Zerlegung.
24     * Wenn kein Algorithmus uebergeben wird (null), dann wird die
25     * Standard-Implementierung des Packages verwendet.
26     * @param algorithm Standard-Algorithmus fuer die QR-Zerlegung.
27     */
28    public static void setDefaultRealQRFactorizationAlgorithm(IRealQRFactorizationAlgorithm
          algorithm) {
29        if (algorithm != _defaultRealQRFactorizationAlgorithm) {
30            if (algorithm != null) {
31                _defaultRealQRFactorizationAlgorithm = algorithm;
32            } else { // if (algorithm == null) ...
33                _defaultRealQRFactorizationAlgorithm = new DefaultRealQRFactorizationAlgorithm
                      ();
34            } // if (algorithm == null) ...
35        } // if (algorithm != _defaultRealQRFactorizationAlgorithm) ...
36    } // public static void setDefaultRealQRFactorizationAlgorithm(
          IRealQRFactorizationAlgorithm algorithm) ...
37
38    /**
39     * Feld, das den Standard-Algorithmus fuer das Loesen linearer Gleichungssysteme speichert.
40     */
41    private static IRealLinearSolverAlgorithm _defaultRealLinearSolverAlgorithm = new
          DefaultRealLinearSolverAlgorithm();
42
43    /**
44     * Gibt den Standard-Algorithmus fuer das Loesen linearer Gleichungssysteme zurueck.
45     * @return Standard-Algorithmus fuer das Loesen linearer Gleichungssysteme.
46     */
47    public static IRealLinearSolverAlgorithm getDefaultRealLinearSolverAlgorithm() {
48        return _defaultRealLinearSolverAlgorithm;
49    } // public static IRealLinearSolverAlgorithm getDefaultRealLinearSolverAlgorithm() ...
50
51    /**
52     * Setzt den Standard-Algorithmus fuer das Loesen linearer Gleichungssysteme.
53     * Wenn kein Algorithmus uebergeben wird (null), dann wird die
54     * Standard-Implementierung des Packages verwendet.
55     * @param algorithm Standard-Algorithmus fuer das Loesen linearer Gleichungssysteme.
56     */
57    public static void setDefaultRealLinearSolverAlgorithm(IRealLinearSolverAlgorithm algorithm
          ) {
58        if (algorithm != _defaultRealLinearSolverAlgorithm) {
59            if (algorithm != null) {
60                _defaultRealLinearSolverAlgorithm = algorithm;
61            } else { // if (algorithm == null) ...
62                _defaultRealLinearSolverAlgorithm = new DefaultRealLinearSolverAlgorithm();
63            } // if (algorithm == null) ...
64        } // if (algorithm != _defaultRealLinearSolverAlgorithm) ...
65    } // public static void setDefaultRealLinearSolverAlgorithm(IRealLinearSolverAlgorithm
          algorithm) ...
66 } // public class DefaultAlgorithms ...
```

Listing 5: Klasse *at.techmath.boltzmann.linearalgebra.DefaultRealLinearSolverAlgorithm*

```
1  package at.techmath.boltzmann.linearalgebra;
2
3  /**
4   * Standard-Loeser fuer lineare Gleichungssysteme.
5   * Verwendet die QR-Zerlegung.
6   * @author Stefan Schnabl (e0226245)
7   */
8  public class DefaultRealLinearSolverAlgorithm implements IRealLinearSolverAlgorithm {
9      /**
10      * QR-Faktorisierungs-Algorithmus, der verwendet werden soll.
11      * Wenn er nicht gesetzt ist, dann wird der DefaultAlgorithm verwendet.
12      */
13     private IRealQRFactorizationAlgorithm _QRAlgorithm = null;
14
15     /**
16      * Gibt den QR-Faktorisierungs-Algorithmus, der verwendet werden soll, zurueck.
17      * Wenn er nicht gesetzt ist, dann wird der DefaultAlgorithm verwendet.
18      * @return QR-Faktorisierungs-Algorithmus, der verwendet werden soll.
19      */
20     public IRealQRFactorizationAlgorithm getQRAlgorithm () {
21         return _QRAlgorithm != null ? _QRAlgorithm : DefaultAlgorithms.
                getDefaultRealQRFactorizationAlgorithm();
22     } // public IRealQRFactorizationAlgorithm getQRAlgorithm() ...
23
24     /**
25      * Setzt den QR-Faktorisierungs-Algorithmus, der verwendet werden soll.
26      * @param qrAlgorithm QR-Faktorisierungs-Algorithmus, der verwendet werden soll.
27      *  Wenn er null ist, dann wird der DefaultAlgorithm verwendet.
28      */
29     public void setQRAlgorithm(IRealQRFactorizationAlgorithm qrAlgorithm) {
30         _QRAlgorithm = qrAlgorithm;
31     } // public void setQRAlgorithm(IRealQRFactorizationAlgorithm qrAlgorithm) ...
32
33     /**
34      * Loest das lineare Gleichnungssystem matrix * Ergebnis = vector.
35      * @param matrix Matrix des Systems.
36      * @param vector Bildvektor des Systems.
37      * @return Loesung der Gleichung matrix * Ergebnis = vector.
38      * @exception java.lang.IllegalArgumentException Wird geworfen, wenn matrix oder vector
            null sind.
39      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Die Dimensionen von
            matrix
40      *  und vector passen nicht zusammen.
41      * @exception at.techmath.boltzmann.linearalgebra.SingularException Die uebergebene Matrix
            ist
42      *  singulaer.
43      */
44     public IRealColumnVector solve(IRealMatrix matrix, IRealColumnVector vector) {
45         if (matrix == null) throw new IllegalArgumentException("matrix = null.");
46         if (vector == null) throw new IllegalArgumentException("vector = null.");
47         if (!matrix.isSquare()) throw new SingularException();
48         int dimension = matrix.getRowCount(); // Matrix ist quadratisch --> RowCount =
               ColumnCount = Dimension.
49         if (dimension != vector.getDimension()) throw new DimensionException();
50         RealColumnVector result = new RealColumnVector(dimension);
51         double[] resultEntries = result._entries;
52         IRealQRFactorization qrFactorization = getQRAlgorithm().factorize(matrix);
53         // Die QR-Faktorisierung liefert eine Darstellung der Matrix A als
54         // A * P = Q * R, wobei P eine Permutationsmatrix, Q eine orthogonale Matrix
55         // und R eine obere Dreiecksmatrix ist.
56         // Um nun das Gleichungssystem A * x = b zu loesen, geht man folgendermassen
57         // vor:
58         // A * x = b <--> A * P * P^-1 * x = b <-->
59         // <--> Q * R * P^-1 * x = b <--> R * P^-1 * x = Q^T b
60         // Man loest also das System R * y = Q^T b, was aufgrund der
61         // oberen Dreiecksform leicht moeglich ist,
62         // und ermittelt dann x = P * y.
63
64         // *) c := Q^T b ermitteln
65         double[] vectorC = new double[dimension];
66         IRealMatrix matrixQ = qrFactorization.getQ();
67         if (matrixQ instanceof RealMatrix) {
68             double[] matrixQEntries = ((RealMatrix) matrixQ)._entries;
69             int currentRow = 0, currentColumn = 0, currentIndex = 0,
70                 maxIndex = matrixQEntries.length;
71             double currentFactor = vector.getEntry(currentRow);
72             while (currentIndex < maxIndex) {
73                 vectorC[currentColumn] += currentFactor * matrixQEntries[currentIndex];
74                 currentColumn++;
```

```
75                  currentIndex++;
76                  if (currentColumn >= dimension) {
77                      currentColumn = 0;
78                      currentRow++;
79                      if (currentRow < dimension) {
80                          currentFactor = vector.getEntry(currentRow);
81                      } // if (currentRow < dimension) ...
82                  } // if (currentColumn >= dimension) ...
83              } // while (currentIndex < maxIndex) ...
84          } else { // if (!(matrixQ instanceof RealMatrix)) ...
85              double currentValue;
86              for (int currentRow = 0; currentRow < dimension; currentRow++) {
87                  currentValue = 0.0D;
88                  for (int currentColumn = 0; currentColumn < dimension; currentColumn++) {
89                      currentValue += matrixQ.getEntry(currentColumn, currentRow) * vector.
                            getEntry(currentColumn);
90                  } // for (int currentColumn = 0; currentColumn < dimension; currentColumn++)
                        ...
91                  vectorC[currentRow] = currentValue;
92              } // for (int currentRow = 0; currentRow < dimension; currentRow++) ...
93          } // if (!(matrixQ instanceof RealMatrix)) ...
94
95          // *) Loesen von R * y = c
96          IRealMatrix matrixR = qrFactorization.getR();
97          if (matrixR instanceof RealMatrix) {
98              double[] matrixREntries = ((RealMatrix) matrixR)._entries;
99              double currentValue;
100             for (int currentRow = dimension - 1; currentRow >= 0; currentRow--) {
101                 currentValue = vectorC[currentRow];
102                 for (int i = currentRow + 1; i < dimension; i++) {
103                     currentValue -= resultEntries[i] * matrixREntries[currentRow * dimension +
                            i];
104                 } // for (int i = currentRow + 1; i < dimension; i++) ...
105                 currentValue /= matrixREntries[currentRow * dimension + currentRow];
106                 if (Double.isInfinite(currentValue) || Double.isNaN(currentValue)) throw new
                        SingularException();
107                 resultEntries[currentRow] = currentValue;
108             } // for (int currentRow = dimension - 1; currentRow >= 0; currentRow--) ...
109         } else { // if (!(matrixR instanceof RealMatrix)) ...
110             double currentValue;
111             for (int currentRow = dimension - 1; currentRow >= 0; currentRow--) {
112                 currentValue = vectorC[currentRow];
113                 for (int i = currentRow + 1; i < dimension; i++) {
114                     currentValue -= resultEntries[i] * matrixR.getEntry(currentRow, i);
115                 } // for (int i = currentRow + 1; i < dimension; i++) ...
116                 currentValue /= matrixR.getEntry(currentRow, currentRow);
117                 if (Double.isInfinite(currentValue) || Double.isNaN(currentValue)) throw new
                        SingularException();
118                 resultEntries[currentRow] = currentValue;
119             } // for (int currentRow = dimension - 1; currentRow >= 0; currentRow--) ...
120         } // if (!(matrixR instanceof RealMatrix)) ...
121
122         // *) Ermitteln von x = P * y
123         return qrFactorization.getP().rightMultiplyColumnVector(result);
124     } // public IRealColumnVector solve(IRealMatrix matrix, IRealColumnVector vector) ...
125 } // public class DefaultRealLinearSolverAlgorithm implements IRealLinearSolverAlgorithm ...
```

Listing 6: Klasse *at.techmath.boltzmann.linearalgebra.DefaultRealQRFactorizationAlgorithm*

```
1  package at.techmath.boltzmann.linearalgebra;
2
3  /**
4   * Standard-Implementierung des QR-Zerlegungs-Algorithmus.
5   * @author Stefan Schnabl (e0226245)
6   */
7  public class DefaultRealQRFactorizationAlgorithm implements IRealQRFactorizationAlgorithm {
8      /**
9       * Implementierung des IRealQRFactorization-Interfaces,
10      * das nur die Matrizen speichert.
11      * @author Stefan Schnabl (e0226245)
12      */
13     private class RealQRFactorizationImpl implements IRealQRFactorization {
14         /**
15          * Feld, das die Permutationsmatrix der Zerlegung speichert.
16          */
17         private IRealPermutationMatrix _P = null;
18
19         /**
20          * Gibt die Permutationsmatrix der Zerlegung zurueck.
21          * @return Permutationsmatrix der Zerlegung.
22          */
23         public IRealPermutationMatrix getP() {
24             return _P;
25         } // public IRealPermutationMatrix getP() ...
26
27         /**
28          * Setzt die Permutationsmatrix der Zerlegung.
29          * @param p Permutationsmatrix der Zerlegung.
30          */
31         public void setP(IRealPermutationMatrix p) {
32             _P = p;
33         } // public void setP(IRealPermutationMatrix p) ...
34
35         /**
36          * Feld, das die orthogonale Matrix der Zerlegung speichert.
37          */
38         private IRealMatrix _Q = null;
39
40         /**
41          * Gibt die orthogonale Matrix der Zerlegung zurueck.
42          * @return Orthogonale Matrix der Zerlegung.
43          */
44         public IRealMatrix getQ() {
45             return _Q;
46         } // public IRealMatrix getQ() ...
47
48         /**
49          * Setzt die orthogonale Matrix der Zerlegung.
50          * @param q Orthogonale Matrix der Zerlegung.
51          */
52         public void setQ(IRealMatrix q) {
53             _Q = q;
54         } // public void setQ(IRealMatrix q) ...
55
56         /**
57          * Feld, das die (verallgemeinerte) obere Dreiecksmatrix
58          * der Zerlegung zurueck.
59          */
60         private IRealMatrix _R = null;
61
62         /**
63          * Gibt die (verallgemeinerte) obere Dreiecksmatrix
64          * der Zerlegung zurueck.
65          * @return Verallgemeinerte obere Dreiecksmatrix der Zerlegung.
66          */
67         public IRealMatrix getR() {
68             return _R;
69         } // public IRealMatrix getR() ...
70
71         /**
72          * Setzt die (verallgemeinerte) obere Dreiecksmatrix
73          * der Zerlegung.
74          * @param r (verallgemeinerte) obere Dreiecksmatrix der Zerlegung.
75          */
76         public void setR(IRealMatrix r) {
77             _R = r;
78         } // public void setR(IRealMatrix r) ...
79     } // private class RealQRFactorizationImpl implements IRealQRFactorization ...
```

```
80
81      /**
82       * Berechnet die QR-Zerlegung der uebergebenen Matrix.
83       * @param matrix Matrix, deren QR-Zerlegung berechnet werden soll.
84       * @return QR-Zerlegung der uebergebenen Matrix.
85       * @exception IllegalArgumentException Wird geworfen, wenn
86       *   die uebergebene Matrix null ist.
87       */
88      public IRealQRFactorization factorize(IRealMatrix matrix) {
89          if (matrix == null) throw new IllegalArgumentException("matrix = null.");
90          if (matrix instanceof RealPermutationMatrix) {
91              // Wenn A eine Permutationsmatrix ist, ist diese orthogonal.
92              // Eine QR-Zerlegung sieht also folgendermassen aus:
93              // A * P = Q * R mit P = I, R = I und Q = A
94              RealQRFactorizationImpl result = new RealQRFactorizationImpl();
95              int dimension = ((RealPermutationMatrix) matrix).getDimension();
96              result.setP(new RealPermutationMatrix(dimension));
97              result.setQ(matrix.getMutableCopy());
98              result.setR(RealDiagonalMatrix.getUnitMatrix(dimension));
99              return result;
100         } else if (matrix instanceof RealDiagonalMatrix) {
101             // Wenn A eine Diagonalmatrix ist, dann ist diese eine obere Dreiecksmatrix.
102             // Eine QR-Zerlegung sieht also folgendermassen aus:
103             // A * P = Q * R mit P = I, Q = I und R = A
104             RealQRFactorizationImpl result = new RealQRFactorizationImpl();
105             int dimension = ((RealDiagonalMatrix) matrix).getDimension();
106             result.setP(new RealPermutationMatrix(dimension));
107             result.setQ(RealDiagonalMatrix.getUnitMatrix(dimension));
108             result.setR(matrix.getMutableCopy());
109             return result;
110         } else { // Allgemeine Matrix ...
111             // Dimensionen ermitteln und Ergebnismatrizen initialisieren.
112             // Es gilt A * P = Q * R, Q und P sind quadratische Matrizen.
113             // --> Dimension von Q muss mit der Zeilenanzahl von A uebereinstimmen,
114             //     Dimension von P muss mit der Spaltenanzahl von A uebereinstimmen,
115             //     R hat gleiche Zeilen- und Spaltenanzahl wie A.
116             // Q wird als Einheitsmatrix initialisiert und durch Linksmultiplikationen
                    aufgebaut.
117             // R wird mit A initialisiert.
118             RealQRFactorizationImpl result = new RealQRFactorizationImpl();
119             // R auf A initialiseren
120             RealMatrix matrixR = new RealMatrix(matrix);
121             double[] matrixEntriesR = matrixR._entries;
122             result.setR(matrixR);
123             int rowCount = matrixR._rowCount;
124             int columnCount = matrixR._columnCount;
125             // Q auf Einheitsmatrix mit Dimension rowCount initialisieren.
126             RealMatrix matrixQ = RealMatrix.getUnitMatrix(rowCount);
127             double[] matrixEntriesQ = matrixQ._entries;
128             result.setQ(matrixQ);
129             // P auf Einheitsmatrix mit Dimension columnCount initialisieren.
130             RealPermutationMatrix matrixP = new RealPermutationMatrix(columnCount);
131             result.setP(matrixP);
132             // Array fuer den Vektor w der Householder-Transformation initialisieren
133             double[] vectorW = new double[rowCount];
134             // Array fuer den Vector v der Householder-Transformation initialisieren
135             // Fuer die Matrix Q hat dieser maximal rowCount Eintraege, da Q eine rowCount x
                    rowCount-Matrix ist,
136             // fuer die Matrix R hat dieser maximal columnCount Eintraege, da es sich um
137             // eine rowCount x columnCount-Matrix handelt, und bei der Berechnung von v
138             // die transponierte Matrix verwendet wird.
139             double[] vectorV = new double[columnCount < rowCount ? rowCount : columnCount];
140             // Spalten-Norm-Quadrate ermitteln
141             double[] columnNormSquare = new double[columnCount];
142             // Die Maximalanzahl der Schritte (falls vorher nicht wegen Restmatrix = 0
                    abgebrochen wird)
143             // ergibt sich als min { rowCount - 1, columnCount }
144             int maxStepCount = rowCount - 1;
145             if (columnCount < maxStepCount) { maxStepCount = columnCount; }
146             // Schrittweise Berechnung der QR-Zerlegung:
147             for (int currentStep = 0; currentStep < maxStepCount; currentStep++) {
148                 // Es wird die (rowCount - currentStep) x (columnCount - currentStep) -
                        Submatrix,
149                 // die in der Zeile und Spalte currentStep beginnt.
150                 // Vorbedingungen vor Schleifendurchlauf:
151                 // .) Die Teilmatrix links der aktuell betrachteten Matrix ist eine
                        verallgemeinerte
152                 //    obere Dreiecksmatrix.
153
154                 // Schritt 1: Berechnen der Normquadrate der Spalten der Submatrix
```

```
155                 {
156                     // Nullsetzen der relevanten Normquadrate.
157                     for (int c = currentStep; c < columnCount; c++) {
158                         columnNormSquare[c] = 0.0D;
159                     } // for (int c = currentStep; c < columnCount; c++) ...
160 
161                     // Berechnen der neuen Normquadrate.
162                     int currentIndex = currentStep * columnCount + currentStep;
163                     int currentColumn = currentStep;
164                     int maxIndex = matrixEntriesR.length;
165                     double currentElement;
166                     while (currentIndex < maxIndex) {
167                         currentElement = matrixEntriesR[currentIndex];
168                         columnNormSquare[currentColumn] += currentElement * currentElement;
169                         currentIndex++;
170                         currentColumn++;
171                         if (currentColumn >= columnCount) {
172                             currentColumn = currentStep;
173                             currentIndex += currentStep;
174                         } // if (currentColumn >= columnCount) ...
175                     } // while (currentIndex < maxIndex) ...
176                 }
177 
178                 // Schritt 2: Wechseln der Spalte mit der groessten Norm mit der neuen erste
                        Spalte.
179                 // Bei diesem Schritt wird gleich ermittelt, ob die Spalte numerisch Null ist
180                 // (Abbruchbedingung, denn wenn die groesste Spalte Null ist, dann die
                        restlichen auch),
181                 // bzw. ob es sich um ein Vielfaches des Einheitsvektors handelt (In diesem
                        Fall
182                 // wird keine Householder-Transformation durchgefuehrt, sondern mit der
                        Einheitsmatrix
183                 // multipliziert, da durch Hinzufuegen der Spalte zur bisher berechneten Matrix
                         die
184                 // obere Dreiecksform erhalten bleibt.
185 
186                 // Spalte mit groesstem Normquadrat ermitteln. Nach Vorbedingung stehen die
187                 // aktuellen Normquadrate im columnNormSquare-Array.
188                 int maxColumnNormSquareIndex = currentStep;
189                 double maxColumnNormSquareNorm = columnNormSquare[currentStep];
190                 for (int currentColumn = currentStep + 1; currentColumn < columnCount;
                        currentColumn++) {
191                     double currentNormSquareNorm = columnNormSquare[currentColumn];
192                     if (currentNormSquareNorm > maxColumnNormSquareNorm) {
193                         maxColumnNormSquareIndex = currentColumn;
194                         maxColumnNormSquareNorm = currentNormSquareNorm;
195                     } // if (currentNormSquareNorm > maxColumnNormSquareNorm) ...
196                 } // for (int currentColumn = currentStep + 1; currentColumn < columnCount;
                        currentColumn++) ...
197 
198                 // Wenn die Maximalspalte nicht die erste Spalte ist, dann Spalten tauschen
199                 // und den Austausch in der Permutationsmatrix protokollieren.
200                 if (maxColumnNormSquareIndex != currentStep) {
201                     int firstIndex, secondIndex;
202                     for (int currentRow = 0; currentRow < rowCount; currentRow++) {
203                         firstIndex = currentRow * columnCount + currentStep;
204                         secondIndex = currentRow * columnCount + maxColumnNormSquareIndex;
205                         double t = matrixEntriesR[firstIndex];
206                         matrixEntriesR[firstIndex] = matrixEntriesR[secondIndex];
207                         matrixEntriesR[secondIndex] = t;
208                     } // for (int currentRow = 0; currentRow < rowCount; currentRow++) ...
209                     matrixP.swapColumns(currentStep, maxColumnNormSquareIndex);
210                 } // if (maxColumnNormSquareIndex != currentStep) ...
211 
212                 // Ermitteln, ob die es sich bei der ersten Spalte um den Nullvektor bzw. ein
213                 // Vielfaches des Einheitsvektors e_currentStep handelt. Normquadrat ermitteln.
214                 double firstEntryNormSquare = 0.0D;
215                 double lowerEntriesNormSquare = 0.0D;
216                 boolean isNullVector = false;
217                 boolean isMultipleOfUnitVector = false;
218                 {
219                   double currentEntry;
220                   for (int currentRow = rowCount - 1; currentRow > currentStep; currentRow--) {
221                       currentEntry = matrixEntriesR[currentRow * columnCount + currentStep];
222                       lowerEntriesNormSquare += currentEntry * currentEntry;
223                   } // for (int currentRow = rowCount - 1; currentRow > currentStep; currentRow
                          --) ...
224                   currentEntry = matrixEntriesR[currentStep * columnCount + currentStep];
225                   firstEntryNormSquare = currentEntry * currentEntry;
226                   if (lowerEntriesNormSquare <= rowCount * 10 * Double.MIN_NORMAL) {
```

```
227                     isMultipleOfUnitVector = true;
228                     if (firstEntryNormSquare < 10 * Double.MIN_NORMAL) {
229                         isNullVector = true;
230                     } // if (firstEntryNormSquare < 10 * Double.MIN_NORMAL) ...
231                 } // if (lowerEntriesNormSquare <= rowCount * 10 * Double.MIN_NORMAL) ...
232             }
233
234             // Schritt 3: Aktuell ersten Vektor durch eine orthogonale Abbildung
235             // auf den jeweiligen Einheitsvektor abbilden.
236             if (isNullVector) {
237                 // Der Vektor mit der maximalen Norm ist der Nullvektor.
238                 // --> Alle restlichen Vektoren sind Nullvektoren.
239                 // --> Die QR-Zerlegung ist fertig, da es sich bei R um
240                 // eine verallgemeinerte Dreiecksmatrix handelt.
241                 break;
242             } else if (isMultipleOfUnitVector) { // && !isNullVector ...
243                 // Es handelt sich bei erstem Vektor um ein Vielfaches des Einheitsvektors.
244                 // --> Die Matrizen werden mit der Einheitsmatrix multipliziert.
245                 // Alle Elemente des ersten Vektors in der Ergebnismatrix ausser das erste
                        auf 0.0 setzen
246                 // Da diese durch Rundungsfehler leicht ungleich 0.0 sein konnten.
247                 for (int currentRow = currentStep + 1; currentRow < rowCount; currentRow++)
                        {
248                     matrixEntriesR[currentRow * columnCount + currentStep] = 0.0D;
249                 } // for (int currentRow = currentStep + 1; currentRow < rowCount;
                        currentRow++) ...
250             } else { // if (!isMultipleOfUnitVector && !isNullVector) ...
251                 // Es handelt sich bei erstem Vektor nicht um ein Vielfaches des
                        Einheitsvektors.
252                 // Es kann eine Householder-Spiegelung durchgefuehrt werden, die den
253                 // ersten Vektor auf den Einheitsvektor abbildet.
254
255                 // Householder-Transformation (Numerik Auzinger S. 80)
256                 // Es sei x aus R^n \ span{e_1} und sigma aus R mit |sigma| = 1
257                 // und sigma * x1 = |x1|. Definiert man
258                 // w := (x + lambda * e_1) / ||x + lambda * e_1||_2 mit lambda := sigma *
                        ||x||_2,
259                 // so gilt ||w||_2 = 1 und Hx = (I - 2ww^T)x = -lambda e_1
260
261                 // Bei der Implementierung fallen Matrix-Matrix-Multiplikationen in der
                        Form
262                 // A^(k+1) = / I_k 0 \ * / U X \ = / U X  \
263                 //           \ 0   H /   \ 0 B /   \ 0 HB /
264                 // an, wobei U aus R^kxk eine obere Dreiecksmatrix ist, X aus K^kx(n-k)
                        eine
265                 // im Allgemeinen vollbesetzte Matrix und B aus R^(m-k)x(n-k) der
                        Matrixblock
266                 // der umgeformt wird. H aus R^(m-k)x(m-k) ist entweder die Identitaet oder
267                 // eine geeignete Householder-Transformation. Gegebenenfalls treten also
268                 // Matrix-Matrix-Multiplikationen vom Typ HB = (I - 2ww^T)B auf, die in der
                        Form
269                 // HB = (I - 2ww^T)B = B - wv^T mit v := 2 B^T w realisiert werden.
270                 // Dieses Vorgehen reduziert den arithmetischen Aufwand erheblich (O(n^3)
                        -> O(n^2)).
271
272                 // Vektor w bestimmen. Er befindet sich im Array w vom Index currentStep
                        bis rowCount - 1.
273                 {
274                     // lambda bestimmen.
275                     double x1 = matrixEntriesR[currentStep * columnCount + currentStep];
276                     double lambda = x1 < 0.0D ?
277                         - Math.sqrt(firstEntryNormSquare + lowerEntriesNormSquare) :
278                         Math.sqrt(firstEntryNormSquare + lowerEntriesNormSquare);
279                     x1 += lambda; // x1 des Vektors x + lambda * e_1 = x1 des Vektors w =
                            lambda
280                     double normW = Math.sqrt(x1*x1 + lowerEntriesNormSquare);
281                     vectorW[currentStep] = x1 / normW;
282                     for (int currentRow = currentStep + 1; currentRow < rowCount;
                            currentRow++) {
283                         vectorW[currentRow] = matrixEntriesR[currentRow * columnCount +
                                currentStep] / normW;
284                     } // for (int currentRow = currentStep + 1; currentRow < rowCount;
                            currentRow++) ...
285                 }
286
287                 // Householder-Transformation auf R anwenden
288                 {
289                     int currentColumn, currentRow, currentIndex, maxIndex;
290                     double currentFactor;
291                     // vT = 2 * wT * B_R bestimmen
```

```
292
293                         // *) Vector nullsetzen
294                         for (currentColumn = currentStep; currentColumn < columnCount;
                                currentColumn++) {
295                             vectorV[currentColumn] = 0.0D;
296                         } // for (currentColumn = currentStep; currentColumn < columnCount;
                                currentColumn++) ...
297
298                         // *) wT * B_R bestimmen
299                         currentRow = currentStep;
300                         currentColumn = currentStep;
301                         currentIndex = currentStep * columnCount + currentStep;
302                         maxIndex = matrixEntriesR.length;
303                         currentFactor = vectorW[currentRow];
304                         while (currentIndex < maxIndex) {
305                             vectorV[currentColumn] += currentFactor * matrixEntriesR[
                                    currentIndex];
306                             currentIndex++;
307                             currentColumn++;
308                             if (currentColumn >= columnCount) {
309                                 currentColumn = currentStep;
310                                 currentIndex += currentStep;
311                                 currentRow++;
312                                 if (currentRow < rowCount) {
313                                     currentFactor = vectorW[currentRow];
314                                 } // if (currentRow < rowCount) ...
315                             } // if (currentColumn >= columnCount) ...
316                         } // while (currentIndex < maxIndex) ...
317
318                         // *) Vector v mit 2 multiplizieren.
319                         for (currentColumn = currentStep; currentColumn < columnCount;
                                currentColumn++) {
320                             vectorV[currentColumn] *= 2.0D;
321                         } // for (currentColumn = currentStep; currentColumn < columnCount;
                                currentColumn++) ...
322
323                         // *) Matrix wvT von B_R subtrahieren.
324                         currentRow = currentStep;
325                         currentColumn = currentStep;
326                         currentIndex = currentStep * columnCount + currentStep;
327                         maxIndex = matrixEntriesR.length;
328                         while (currentIndex < maxIndex) {
329                             matrixEntriesR[currentIndex] -= vectorW[currentRow] * vectorV[
                                    currentColumn];
330                             currentIndex++;
331                             currentColumn++;
332                             if (currentColumn >= columnCount) {
333                                 currentColumn = currentStep;
334                                 currentIndex += currentStep;
335                                 currentRow++;
336                             } // if (currentColumn >= columnCount) ...
337                         } // while (currentIndex < maxIndex) ...
338                     }
339
340                     // Householder-Transformation auf Q anwenden
341                     {
342                         // Die Matrix Q ergibt sich durch iterative Multiplikation von rechts
                                mit
343                         // der Householder-Transformation des aktuellen Schritts.
344                         //
345                         // Q = H1 * H2 * ... * Hk k .. Anzahl der Gesamtschritte.
346                         // Es kommen also jeweils Multiplikationen der Form (n = rowCount)
347                         // (Q_(n)x(k-1) Q_(n)x(n-k+1)) * / I_(k-1) 0  \ = (Q_(n)x(k-1) U*Hk)
348                         //                    =:U          \ 0       Hk /
349                         // U * Hk = U * (I_(n-k+1) - 2wwT) = U - 2(Uw)wT
350                         // Die Multiplikation im aktuellen Schritt kann also durchgefuehrt
                                werden,
351                         // indem man 2 * v wT von der Matrix subtrahiert, mit v = Bw
352
353                         int currentColumn, currentRow, currentIndex, maxIndex;
354
355                         // *) Vektor v nullsetzen
356                         for (currentRow = 0; currentRow < rowCount; currentRow++) {
357                             vectorV[currentRow] = 0.0D;
358                         } // for (currentRow = 0; currentRow < rowCount; currentRow++) ...
359
360                         // *) v = U*w bestimmen
361                         currentRow = 0;
362                         currentColumn = currentStep;
363                         currentIndex = currentStep;
```

```
364                         maxIndex = matrixEntriesQ.length;
365                         while (currentIndex < maxIndex) {
366                             vectorV[currentRow] += vectorW[currentColumn] * matrixEntriesQ[
                                    currentIndex];
367                             currentIndex++;
368                             currentColumn++;
369                             if (currentColumn >= rowCount) {
370                                 currentColumn = currentStep;
371                                 currentIndex += currentStep;
372                                 currentRow++;
373                             } // if (currentColumn >= rowCount) ...
374                         } // while (currentIndex < maxIndex) ...
375
376                         // *) Vektor v mit 2 multiplizieren
377                         for (currentRow = 0; currentRow < rowCount; currentRow++) {
378                             vectorV[currentRow] *= 2.0D;
379                         } // for (currentRow = 0; currentRow < rowCount; currentRow++) ...
380
381                         // *) Matrix 2 * v wT von Q subtrahieren
382                         currentRow = 0;
383                         currentColumn = currentStep;
384                         currentIndex = currentStep;
385                         maxIndex = matrixEntriesQ.length;
386                         while (currentIndex < maxIndex) {
387                             matrixEntriesQ[currentIndex] -= vectorV[currentRow] * vectorW[
                                    currentColumn];
388                             currentIndex++;
389                             currentColumn++;
390                             if (currentColumn >= rowCount) {
391                                 currentColumn = currentStep;
392                                 currentIndex += currentStep;
393                                 currentRow++;
394                             } // if (currentColumn >= rowCount) ...
395                         } // while (currentIndex < maxIndex) ...
396                     }
397
398                     // Alle Elemente des ersten Vektors in der Ergebnismatrix ausser das erste
                            auf 0.0 setzen
399                     // Da diese durch Rundungsfehler leicht ungleich 0.0 sein konnten.
400                     for (int currentRow = currentStep + 1; currentRow < rowCount; currentRow++)
                            {
401                         matrixEntriesR[currentRow * columnCount + currentStep] = 0.0D;
402                     } // for (int currentRow = currentStep + 1; currentRow < rowCount;
                            currentRow++) ...
403                 } // if (!isMultipleOfUnitVector && !isNullVector) ...
404             } // for (int currentStep = 0; currentStep < maxStepCount; currentStep++) ...
405             return result;
406         } // Allgemeine Matrix ...
407     } // public IRealQRFactorization factorize(IRealMatrix matrix) ...
408 } // public class DefaultRealQRFactorizationAlgorithm implements IRealQRFactorizationAlgorithm
        ...
```

Listing 7: Klasse *at.techmath.boltzmann.linearalgebra.DimensionException*

```
1  package at.techmath.boltzmann.linearalgebra;
2
3  /**
4   * Exception, die geworfen wird, wenn eine ungueltige Dimension bzw. Dimensions-Kombination
5   * aufgetreten ist.
6   * @author Stefan Schnabl (e0226245)
7   */
8  public class DimensionException extends RuntimeException {
9      private static final long serialVersionUID = -7546269446363770713L;
10     public DimensionException() { super(); }
11     public DimensionException(String message) { super(message); }
12     public DimensionException(String message, Throwable cause) { super(message, cause); }
13     public DimensionException(Throwable cause) { super(cause); }
14 } // public class DimensionException extends RuntimeException ...
```

Listing 8: Interface *at.techmath.boltzmann.linearalgebra.IMutableRealColumnVector*

```
1  package at.techmath.boltzmann.linearalgebra;
2
3  /**
4   * Interface, das einen veraenderbaren reellwertigen Spaltenvektor darstellt.
5   * @author Stefan Schnabl (e0226245)
6   */
7  public interface IMutableRealColumnVector extends IRealColumnVector, IMutableRealVector {
8      /**
9       * Addiert den uebergebenen Vektor zum aktuellen Vektor.
10      * Wenn der uebergebene Vektor null ist, wird der Nullvektor addiert.
11      * @param vector Vektor, der addiert werden soll.
12      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Wird geworfen, wenn
            die Dimension
13      *   des uebergebenen Vektors nicht mit der des aktuellen Vektors uebereinstimmt.
14      */
15     void addVector(IRealColumnVector vector);
16
17     /**
18      * Addiert das factor-Fache des uebergebenen Vektors zum aktuellen Vektor.
19      * Wenn der uebergebene Vektor null ist, wird der Nullvektor addiert.
20      * @param vector Vektor, dessen factor-Faches addiert werden soll.
21      * @param factor Faktor, mit dem der uebergebene Vektor multipliziert werden soll.
22      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Wird geworfen, wenn
            die Dimension
23      *   des uebergebenen Vektors nicht mit der des aktuellen Vektors uebereinstimmt.
24      */
25     void addVector(IRealColumnVector vector, double factor);
26
27     /**
28      * Subtrahiert den uebergebenen Vektor vom aktuellen Vektor.
29      * Wenn der uebergebene Vektor null ist, wird der Nullvektor subtrahiert.
30      * @param vector Vektor, der subtrahiert werden soll.
31      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Wird geworfen, wenn
            die Dimension
32      *   des uebergebenen Vektors nicht mit der des aktuellen Vektors uebereinstimmt.
33      */
34     void subtractVector(IRealColumnVector vector);
35
36     /**
37      * Subtrahiert das factor-Fache des uebergebenen Vektors vom aktuellen Vektor.
38      * Wenn der uebergebene Vektor null ist, wird der Nullvektor subtrahiert.
39      * @param vector Vektor, dessen factor-Faches subtrahiert werden soll.
40      * @param factor Faktor, mit dem der uebergebene Vektor multipliziert werden soll.
41      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Wird geworfen, wenn
            die Dimension
42      *   des uebergebenen Vektors nicht mit der des aktuellen Vektors uebereinstimmt.
43      */
44     void subtractVector(IRealColumnVector vector, double factor);
45 } // public interface IMutableRealColumnVector extends IRealColumnVector, IMutableRealVector
      ...
```

Listing 9: Interface *at.techmath.boltzmann.linearalgebra.IMutableRealMatrix*

```
1  package at.techmath.boltzmann.linearalgebra;
2
3  /**
4   * Interface, das eine reellwertige Matrix darstellt, bei der jeder Eintrag
5   * veraendert werden kann.
6   * @author Stefan Schnabl (e0226245)
7   */
8  public interface IMutableRealMatrix extends IRealMatrix {
9      /**
10      * Setzt den Eintrag an der angegebenen Position
11      * auf den angegebenen Wert.
12      * @param row Zeile des Eintrags (Nullbasiert).
13      * @param column Spalte des Eintrags (Nullbasiert).
14      * @param value Wert, auf den der Eintrag gesetzt werden soll.
15      * @exception IndexOutOfBoundsException Wird geworfen, wenn die
16      *   angegebene Position nicht existiert.
17      */
18     void setEntry(int row, int column, double value);
19
20     /**
21      * Addiert zum Eintrag an der angegebenen Position
22      * den angegebenen Wert.
23      * @param row Zeile des Eintrags (Nullbasiert).
24      * @param column Spalte des Eintrags (Nullbasiert).
25      * @param value Wert, der addiert werden soll.
26      * @exception IndexOutOfBoundsException Wird geworfen, wenn die
27      *   angegebene Position nicht existiert.
28      */
29     void addEntry(int row, int column, double value);
30
31     /**
32      * Multipliziert den Eintrag an der angegebenen Position
33      * mit dem angegebenen Faktor.
34      * @param row Zeile des Eintrags (Nullbasiert).
35      * @param column Spalte des Eintrags (Nullbasiert).
36      * @param factor Faktor, mit dem der Eintrag multipliziert werden soll.
37      * @exception IndexOutOfBoundsException Wird geworfen, wenn die
38      *   angegebene Position nicht existiert.
39      */
40     void multiplyEntry(int row, int column, double factor);
41 } // public interface IMutableRealMatrix extends IRealMatrix ...
```

Listing 10: Interface *at.techmath.boltzmann.linearalgebra.IMutableRealRowVector*

```
1  package at.techmath.boltzmann.linearalgebra;
2
3  /**
4   * Interface, das einen veraenderbaren reellwertigen Zeilenvektor darstellt.
5   * @author Stefan Schnabl (e0226245)
6   */
7  public interface IMutableRealRowVector extends IRealRowVector, IMutableRealVector {
8      /**
9       * Addiert den uebergebenen Vektor zum aktuellen Vektor.
10      * Wenn der uebergebene Vektor null ist, wird der Nullvektor addiert.
11      * @param vector Vektor, der addiert werden soll.
12      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Wird geworfen, wenn
            die Dimension
13      *   des uebergebenen Vektors nicht mit der des aktuellen Vektors uebereinstimmt.
14      */
15     void addVector(IRealRowVector vector);
16
17     /**
18      * Addiert das factor-Fache des uebergebenen Vektors zum aktuellen Vektor.
19      * Wenn der uebergebene Vektor null ist, wird der Nullvektor addiert.
20      * @param vector Vektor, dessen factor-Faches addiert werden soll.
21      * @param factor Faktor, mit dem der uebergebene Vektor multipliziert werden soll.
22      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Wird geworfen, wenn
            die Dimension
23      *   des uebergebenen Vektors nicht mit der des aktuellen Vektors uebereinstimmt.
24      */
25     void addVector(IRealRowVector vector, double factor);
26
27     /**
28      * Subtrahiert den uebergebenen Vektor vom aktuellen Vektor.
29      * Wenn der uebergebene Vektor null ist, wird der Nullvektor subtrahiert.
30      * @param vector Vektor, der subtrahiert werden soll.
31      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Wird geworfen, wenn
            die Dimension
32      *   des uebergebenen Vektors nicht mit der des aktuellen Vektors uebereinstimmt.
33      */
34     void subtractVector(IRealRowVector vector);
35
36     /**
37      * Subtrahiert das factor-Fache des uebergebenen Vektors vom aktuellen Vektor.
38      * Wenn der uebergebene Vektor null ist, wird der Nullvektor subtrahiert.
39      * @param vector Vektor, dessen factor-Faches subtrahiert werden soll.
40      * @param factor Faktor, mit dem der uebergebene Vektor multipliziert werden soll.
41      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Wird geworfen, wenn
            die Dimension
42      *   des uebergebenen Vektors nicht mit der des aktuellen Vektors uebereinstimmt.
43      */
44     void subtractVector(IRealRowVector vector, double factor);
45 } // public interface IMutableRealRowVector extends IRealRowVector, IMutableRealVector ...
```

Listing 11: Interface *at.techmath.boltzmann.linearalgebra.IMutableRealVector*

```
1  package at.techmath.boltzmann.linearalgebra;
2
3  /**
4   * Interface, das einen veraenderbaren reellwertigen Vektor darstellt.
5   * @author Stefan Schnabl (e0226245)
6   */
7  public interface IMutableRealVector extends IRealVector {
8      /**
9       * Setzt den Eintrag mit dem angegebenen Index auf den angegebenen Wert.
10      * @param index Index, dessen Eintrag gesetzt werden soll (Nullbasiert).
11      * @param value Wert, auf den der Eintrag gesetzt werden soll.
12      * @exception IndexOutOfBoundsException Wird geworfen, wenn der
13      *   Index < 0 oder >= Dimension ist.
14      */
15     void setEntry(int index, double value);
16
17     /**
18      * Addiert den angegebenen Wert zum Eintrag mit dem angegebenen Index.
19      * @param index Index des Eintrags.
20      * @param value Wert, der zum Eintrag addiert werden soll.
21      * @exception IndexOutOfBoundsException Wird geworfen, wenn der
22      *   Index < 0 oder >= Dimension ist.
23      */
24     void addEntry(int index, double value);
25
26     /**
27      * Multipliziert den Eintrag mit dem angegebenen Index mit dem angegebenen Faktor.
28      * @param index Index des Eintrags.
29      * @param factor Faktor, mit dem der Eintrag multipliziert werden soll.
30      * @exception IndexOutOfBoundsException Wird geworfen, wenn der
31      *   Index < 0 oder >= Dimension ist.
32      */
33     void multiplyEntry(int index, double factor);
34
35     /**
36      * Multipliziert den aktuellen Vektor mit dem uebergebenen Faktor.
37      * @param factor Faktor, mit dem der Vektor multipliziert werden soll.
38      */
39     void scalarMultiply(double factor);
40 } // public interface IMutableRealVector extends IRealVector ...
```

Listing 12: Interface *at.techmath.boltzmann.linearalgebra.IRealColumnVector*

```
1  package at.techmath.boltzmann.linearalgebra;
2
3  /**
4   * Interface, das einen reellwertigen Spaltenvektor repraesentiert.
5   * @author Stefan Schnabl (e0226245)
6   */
7  public interface IRealColumnVector extends IRealVector {
8      /**
9       * Gibt eine Kopie des transponierten Vektors zurueck.
10      * @return Kopie des transponierten Vektors.
11      */
12     IRealRowVector transpose();
13
14     /**
15      * Ermittelt das kanonische Skalarprodukt des aktuellen Vektors mit dem
16      * uebergebenen Vektor.
17      * @param vector Vektor, mit dem das kanonische Skalarprodukt gebildet werden soll.
18      *   Wenn vector = null ist, wird mit dem Nullvektor multipliziert.
19      * @return Kanonisches Skalarprodukt des aktuellen Vektors mit dem uebergebenen Vektor.
20      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Dimension des
            uebergebenen Vektors stimmt
21      *   nicht mit der Dimension des aktuellen Vektors ueberein.
22      */
23     double getInnerProduct(IRealColumnVector vector);
24 } // public interface IRealColumnVector extends IRealVector ...
```

Listing 13: Interface *at.techmath.boltzmann.linearalgebra.IRealDiagonalMatrix*

```
1  package at.techmath.boltzmann.linearalgebra;
2
3  /**
4   * Repraesentiert eine reellwertige Diagonalmatrix.
5   * @author Stefan Schnabl (e0226245)
6   */
7  public interface IRealDiagonalMatrix extends IRealMatrix {
8      /**
9       * Gibt die Dimension der Matrix zurueck.
10      * @return Dimension der Matrix.
11      */
12     int getDimension();
13
14     /**
15      * Gibt den Diagonaleintrag mit dem angegebenen Index zurueck.
16      * @param index Gewuenschter Index.
17      * @return Diagonaleintrag mit dem angegebenen Index (Nullbasiert).
18      * @exception IndexOutOfBoundsException Wird geworfen, wenn der uebergebene
19      *   Index nicht existiert.
20      */
21     double getDiagonalEntry(int index);
22
23     /**
24      * Setzt den Diagonaleintrag mit dem angegebenen Index
25      * auf den angegebenen Wert.
26      * @param index Index des Eintrags, der gesetzt werden soll (Nullbasiert).
27      * @param value Wert, auf den der Eintrag gesetzt werden soll.
28      * @exception IndexOutOfBoundsException Wird geworfen, wenn der uebergebene
29      *   Index nicht existiert.
30      */
31     void setDiagonalEntry(int index, double value);
32
33     /**
34      * Addiert zum Diagonaleintrag mit dem angegebenen Index
35      * den angegebenen Wert.
36      * @param index Index des Eintrags, zu dem der Wert addiert werden soll (Nullbasiert).
37      * @param value Wert, der addiert werden soll.
38      * @exception IndexOutOfBoundsException Wird geworfen, wenn der uebergebene
39      *   Index nicht existiert.
40      */
41     void addDiagonalEntry(int index, double value);
42
43     /**
44      * Multipliziert den Diagonaleintrag mit dem angegebenen Index
45      * mit dem angegebenen Faktor.
46      * @param index Index des Eintrags, der mit dem Faktor multipliziert werden soll (
            Nullbasiert).
47      * @param factor Faktor, mit dem multipliziert werden soll.
48      * @exception IndexOutOfBoundsException Wird geworfen, wenn der uebergebene
49      *   Index nicht existiert.
50      */
51     void multiplyDiagonalEntry(int index, double factor);
52 } // public interface IRealDiagonalMatrix extends IRealMatrix ...
```

Listing 14: Interface *at.techmath.boltzmann.linearalgebra.IRealLinearSolverAlgorithm*

```
1  package at.techmath.boltzmann.linearalgebra;
2
3   /**
4    * Algorithmus der ein lineares Gleichungssystem der Form A * x = b loest.
5    * @author Stefan Schnabl (e0226245)
6    */
7  public interface IRealLinearSolverAlgorithm {
8      /**
9       * Loest das lineare Gleichnungssystem matrix * Ergebnis = vector.
10      * @param matrix Matrix des Systems.
11      * @param vector Bildvektor des Systems.
12      * @return Loesung der Gleichung matrix * Ergebnis = vector.
13      * @exception IllegalArgumentException Wird geworfen, wenn matrix oder vector null sind.
14      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Die Dimensionen von
            matrix
15      *  und vector passen nicht zusammen.
16      * @exception at.techmath.boltzmann.linearalgebra.SingularException Die uebergebene Matrix
            ist
17      *  singulaer.
18      */
19     IRealColumnVector solve(IRealMatrix matrix, IRealColumnVector vector);
20 } // public interface IRealLinearSolverAlgorithm ...
```

Listing 15: Interface *at.techmath.boltzmann.linearalgebra.IRealMatrix*

```
1  package at.techmath.boltzmann.linearalgebra;
2
3  /**
4   * Interface, das eine reellwertige Matrix repraesentiert.
5   * @author Stefan Schnabl (e0226245)
6   */
7  public interface IRealMatrix {
8      /**
9       * Gibt die Anzahl der Zeilen der Matrix zurueck.
10      * @return Anzahl der Zeilen der Matrix.
11      */
12     int getRowCount();
13
14     /**
15      * Gibt die Anzahl der Spalten der Matrix zurueck.
16      * @return Anzahl der Spalten der Matrix.
17      */
18     int getColumnCount();
19
20     /**
21      * Gibt zurueck, ob es sich um eine quadratische Matrix handelt.
22      * @return true, wenn es sich um eine quadratische Matrix handelt, ansonsten false.
23      */
24     boolean isSquare();
25
26     /**
27      * Gibt den Eintrag an der angegebenen Position zurueck.
28      * @param row Zeile des Eintrags (Nullbasiert).
29      * @param column Spalte des Eintrags (Nullbasiert).
30      * @return Eintrag an der angegebenen Position.
31      * @exception IndexOutOfBoundsException Wird geworfen, wenn die angegebene
32      *   Position nicht existiert.
33      */
34     double getEntry(int row, int column);
35
36     /**
37      * Gibt eine veraenderbare Kopie der aktuellen Matrix zurueck.
38      * @return Veraenderbare Kopie der aktuellen Matrix.
39      */
40     IMutableRealMatrix getMutableCopy();
41
42     /**
43      * Multipliziert die Matrix von links mit dem angegebenen Zeilenvektor
44      * und gibt das Ergebnis als veraenderbaren Vektor zurueck.
45      * Die aktuelle Matrix bleibt unangetastet.
46      * @param vector Vektor, mit dem die Matrix multipliziert werden soll.
47      *   (null wird als Nullvektor interpraetiert)
48      * @return Ergebnisvektor.
49      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Wird geworfen, wenn
            die Dimensionen
50      *   nicht uebereinstimmen.
51      */
52     IMutableRealRowVector leftMultiplyRowVector(IRealRowVector vector);
53
54     /**
55      * Multipliziert die Matrix von rechts mit dem angegebenen Zeilenvektor
56      * und gibt das Ergebnis als Matrix zurueck. Die aktuelle Matrix bleibt
57      * unangetastet.
58      * @param vector Vektor, mit dem die Matrix multipliziert werden soll.
59      * @return Ergebnismatrix.
60      * @exception IllegalArgumentException Wird geworfen, wenn der angegebene Vektor null ist.
61      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Wird geworfen, wenn
            die Dimensionen
62      *   nicht uebereinstimmen.
63      */
64     IRealMatrix rightMultiplyRowVector(IRealRowVector vector);
65
66     /**
67      * Multipliziert die Matrix von links mit dem angegebenen Spaltenvektor
68      * und gibt das Ergebnis als Matrix zurueck. Die aktuelle Matrix bleibt
69      * unangetastet.
70      * @param vector Vektor, mit dem die Matrix multipliziert werden soll.
71      * @return Ergebnismatrix.
72      * @exception IllegalArgumentException Wird geworfen, wenn der angegebene Vektor null ist.
73      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Wird geworfen, wenn
            die Dimensionen
74      *   nicht uebereinstimmen.
75      */
76     IRealMatrix leftMultiplyColumnVector(IRealColumnVector vector);
```

```
77
78      /**
79       * Multipliziert die Matrix von rechts mit dem angegebenen Spaltenvektor
80       * und gibt das Ergebnis als veraenderbaren Vektor zurueck.
81       * Die aktuelle Matrix bleibt unangetastet.
82       * @param vector Vektor, mit dem die Matrix multipliziert werden soll.
83       *   (null wird als Nullvektor interpraetiert)
84       * @return Ergebnisvektor.
85       * @exception at.techmath.boltzmann.linearalgebra.DimensionException Wird geworfen, wenn
             die Dimensionen
86       *   nicht uebereinstimmen.
87       */
88      IMutableRealColumnVector rightMultiplyColumnVector(IRealColumnVector vector);
89
90      /**
91       * Multipliziert die Matrix von links mit der angegebenen Matrix
92       * und gibt das Ergebnis als Matrix zurueck. Die aktuelle Matrix bleibt
93       * unangetastet.
94       * @param matrix Matrix, mit der multipliziert werden soll.
95       * @return Ergebnismatrix.
96       * @exception IllegalArgumentException Wird geworfen, wenn die uebergebene Matrix null ist.
97       * @exception at.techmath.boltzmann.linearalgebra.DimensionException Wird geworfen, wenn
             die Dimensionen
98       *   nicht uebereinstimmen.
99       */
100     IRealMatrix leftMultiplyMatrix(IRealMatrix matrix);
101
102     /**
103      * Multipliziert die Matrix von rechts mit der angegebenen Matrix
104      * und gibt das Ergebnis als Matrix zurueck. Die aktuelle Matrix bleibt
105      * unangetastet.
106      * @param matrix Matrix, mit der multipliziert werden soll.
107      * @return Ergebnismatrix.
108      * @exception IllegalArgumentException Wird geworfen, wenn die uebergebene Matrix null ist.
109      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Wird geworfen, wenn
             die Dimensionen
110      *   nicht uebereinstimmen.
111      */
112     IRealMatrix rightMultiplyMatrix(IRealMatrix matrix);
113 } // public interface IRealMatrix ...
```

Listing 16: Interface *at.techmath.boltzmann.linearalgebra.IRealPermutationMatrix*

```
1  package at.techmath.boltzmann.linearalgebra;
2
3  /**
4   * Interface, das eine reellwertige Permutationsmatrix darstellt.
5   * @author Stefan Schnabl (e0226245)
6   */
7  public interface IRealPermutationMatrix extends IRealMatrix {
8      /**
9       * Tauscht die beiden angegebenen Spalten aus.
10      * @param firstColumn Index der ersten Spalte.
11      * @param secondColumn Index der zweiten Spalte.
12      * @exception IndexOutOfBoundsException Wird geworfen, wenn mindestens
13      *   einer der beiden angegebenen Indizes ausserhalb des gueltigen Bereichs ist.
14      */
15     void swapColumns(int firstColumn, int secondColumn);
16
17     /**
18      * Tauscht die beiden angegebenen Zeilen aus.
19      * @param firstRow Index der ersten Zeile.
20      * @param secondRow Index der zweiten Zeile.
21      * @exception IndexOutOfBoundsException Wird geworfen, wenn mindestens
22      *   einer der beiden angegebenen Indizes ausserhalb des gueltigen Bereichs ist.
23      */
24     void swapRows(int firstRow, int secondRow);
25
26     /**
27      * Gibt die Dimension der Matrix zurueck.
28      * @return Dimension der Matrix.
29      */
30     int getDimension();
31 } // public interface IRealPermutationMatrix extends IRealMatrix ...
```

Listing 17: Interface *at.techmath.boltzmann.linearalgebra.IRealQRFactorization*

```
1  package at.techmath.boltzmann.linearalgebra;
2
3  /**
4   * Reellwertige QR-Faktorisierung einer Matrix A in folgender Form:
5   * A*P = Q*R, wobei P eine Permutationsmatrix,
6   * Q eine orthogonale Matrix und R eine (verallgemeinerte)
7   * obere Dreiecksmatrix ist.
8   * @author Stefan Schnabl (e0226245)
9   */
10 public interface IRealQRFactorization {
11     /**
12      * Gibt die Permutationsmatrix der Zerlegung zurueck.
13      * @return Permutationsmatrix der Zerlegung.
14      */
15     IRealPermutationMatrix getP();
16
17     /**
18      * Gibt die orthogonale Matrix der Zerlegung zurueck.
19      * @return Orthogonale Matrix der Zerlegung.
20      */
21     IRealMatrix getQ();
22
23     /**
24      * Gibt die (verallgemeinerte) obere Dreiecksmatrix
25      * der Zerlegung zurueck.
26      * @return Verallgemeinerte obere Dreiecksmatrix der Zerlegung.
27      */
28     IRealMatrix getR();
29 } // public interface IRealQRFactorization ...
```

Listing 18: Interface *at.techmath.boltzmann.linearalgebra.IRealQRFactorizationAlgorithm*

```
1  package at.techmath.boltzmann.linearalgebra;
2
3  /**
4   * Algorithmus der die reellwertige QR-Zerlegung berechnet.
5   * @author Stefan Schnabl (e0226245)
6   */
7  public interface IRealQRFactorizationAlgorithm {
8      /**
9       * Berechnet die QR-Zerlegung der uebergebenen Matrix.
10      * @param matrix Matrix, deren QR-Zerlegung berechnet werden soll.
11      * @return QR-Zerlegung der uebergebenen Matrix.
12      * @exception IllegalArgumentException Wird geworfen, wenn
13      *   die uebergebene Matrix null ist.
14      */
15     IRealQRFactorization factorize(IRealMatrix matrix);
16 } // public interface IRealQRFactorizationAlgorithm ...
```

Listing 19: Interface *at.techmath.boltzmann.linearalgebra.IRealRowVector*

```
1  package at.techmath.boltzmann.linearalgebra;
2
3  /**
4   * Interface, das einen reellwertigen Zeilenvektor repraesentiert.
5   * @author Stefan Schnabl (e0226245)
6   */
7  public interface IRealRowVector extends IRealVector {
8      /**
9       * Gibt eine Kopie des transponierten Vektors zurueck.
10      * @return Kopie des transponierten Vektors.
11      */
12     IRealColumnVector transpose();
13
14     /**
15      * Ermittelt das kanonische Skalarprodukt des aktuellen Vektors mit dem
16      * uebergebenen Vektor.
17      * @param vector Vektor, mit dem das kanonische Skalarprodukt gebildet werden soll.
18      *   Wenn vector = null ist, wird mit dem Nullvektor multipliziert.
19      * @return Kanonisches Skalarprodukt des aktuellen Vektors mit dem uebergebenen Vektor.
20      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Dimension des
            uebergebenen Vektors stimmt
21      *   nicht mit der Dimension des aktuellen Vektors ueberein.
22      */
23     double getInnerProduct(IRealRowVector vector);
24 } // public interface IRealRowVector extends IRealVector ...
```

Listing 20: Interface *at.techmath.boltzmann.linearalgebra.IRealVector*

```
1  package at.techmath.boltzmann.linearalgebra;
2
3  /**
4   * Interface, das einen reelwertigen Vektor repraesentiert.
5   * @author Stefan Schnabl (e0226245)
6   */
7  public interface IRealVector {
8      /**
9       * Gibt die Dimension des Vektors zurueck.
10      * @return Dimension des Vektors.
11      */
12     int getDimension();
13
14     /**
15      * Gibt den Eintrag mit dem angegebenen Index zurueck.
16      * @param index Index, dessen Eintrag zurueckgegeben werden soll (Nullbasiert).
17      * @return Eintrag mit dem angegebenen Index.
18      * @exception IndexOutOfBoundsException Wird geworfen, wenn der
19      *   Index < 0 oder >= Dimension ist.
20      */
21     double getEntry(int index);
22
23     /**
24      * Ermittelt die Maximumsnorm des aktuellen Vektors.
25      * @return Maximumsnorm des aktuellen Vektors.
26      */
27     double getNormMax();
28
29     /**
30      * Ermittelt die l1-Norm des aktuellen Vektors.
31      * @return l1-Norm des aktuellen Vektors.
32      */
33     double getNormSum();
34
35     /**
36      * Ermittelt die euklidische Norm des aktuellen Vektors.
37      * @return Euklidische Norm des aktuellen Vektors.
38      */
39     double getNorm();
40 } // public interface IRealVector ...
```

Listing 21: Klasse *at.techmath.boltzmann.linearalgebra.RealColumnVector*

```
1  package at.techmath.boltzmann.linearalgebra;
2
3  /**
4   * Die RealColumnVector-Klasse stellt einen reellwertigen Spaltenvektor dar.
5   * @author Stefan Schnabl (e0226245)
6   */
7  public class RealColumnVector implements IMutableRealColumnVector {
8      /**
9       * Erstellt eine neue RealColumnVector-Instanz, die
10      * mit 0 initialisiert ist, und die angegebene Dimension besitzt.
11      * @param dimension Dimension des Vektors.
12      * @exception at.techmath.boltzmann.linearalgebra.DimensionException
13      *   Wird geworfen, wenn dimension < 1 ist.
14      */
15     public RealColumnVector(int dimension) {
16         if (dimension < 1) throw new DimensionException();
17         _entries = new double[dimension];
18         for (int c = 0; c < _entries.length; c++) {
19             _entries[c] = 0.0D;
20         } // for (int c = 0; c < _entries.length; c++) ...
21     } // public RealColumnVector(int dimension) ...
22
23     /**
24      * Erstellt eine neue RealColumnVector-Instanz, die mit
25      * dem uebergebenen Array initialisiert wird.
26      * @param entries Array der Eintraege des Vektors.
27      * @exception IllegalArgumentException Wird geworfen,
28      *   wenn entries null.
29      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Wird geworfen,
30      *   wenn entries.length < 1 ist.
31      */
32     public RealColumnVector(double[] entries) {
33         if (entries == null) throw new IllegalArgumentException("entries = null.");
34         int dimension = entries.length;
35         if (dimension < 1) throw new DimensionException("entries.length < 1.");
36         _entries = new double[dimension];
37         for (int c = 0; c < dimension; c++) {
38             _entries[c] = entries[c];
39         } // for (int c = 0; c < dimension; c++) ...
40     } // public RealColumnVector(double[] entries) ...
41
42     /**
43      * Erstellt eine Kopie der angegebenen IRealColumnVector-Instanz.
44      * @param source IRealColumnVector-Instanz, die kopiert werden soll.
45      * @exception IllegalArgumentException Wird geworfen, wenn source = null ist.
46      */
47     public RealColumnVector(IRealColumnVector source) {
48         if (source == null) throw new IllegalArgumentException("source = null.");
49         int dimension = source.getDimension();
50         _entries = new double[dimension];
51         for (int c = 0; c < dimension; c++) {
52             _entries[c] = source.getEntry(c);
53         } // for (int c = 0; c < dimension; c++) ...
54     } // public RealColumnVector(IRealColumnVector source) ...
55
56     /**
57      * Gibt eine String-Repraesentation des Vektors zurueck.
58      * @return String-Repraesentation des Vektors.
59      */
60     @Override
61     public String toString() {
62         int dimension = _entries.length;
63         StringBuilder sb = new StringBuilder();
64         sb.append('(');
65         for (int c = 0; c < dimension; c++) {
66             if (c > 0) {
67                 sb.append(',');
68             } // if (c > 0) ...
69             sb.append(_entries[c]);
70         } // for (int c = 0; c < dimension; c++) ...
71         sb.append(")T");
72         return sb.toString();
73     } // public String toString() ...
74
75     /**
76      * Array der Eintraege des Vektors.
77      */
78     protected double[] _entries;
79
```

```
80      /**
81       * Gibt die Dimension des Vektors zurueck.
82       * @return Dimension des Vektors.
83       */
84      public int getDimension() {
85          return _entries.length;
86      } // public int getDimension() ...
87
88      /**
89       * Gibt den Eintrag mit dem angegebenen Index zurueck.
90       * @param index Index, dessen Eintrag zurueckgegeben werden soll (Nullbasiert).
91       * @return Eintrag mit dem angegebenen Index.
92       * @exception IndexOutOfBoundsException Wird geworfen, wenn der
93       *   Index < 0 oder >= Dimension ist.
94       */
95      public double getEntry(int index) {
96          if (index < 0 || index >= _entries.length) throw new IndexOutOfBoundsException();
97          return _entries[index];
98      } // public double getEntry(int index) ...
99
100     /**
101      * Setzt den Eintrag mit dem angegebenen Index auf den angegebenen Wert.
102      * @param index Index, dessen Eintrag gesetzt werden soll (Nullbasiert).
103      * @param value Wert, auf den der Eintrag gesetzt werden soll.
104      * @exception IndexOutOfBoundsException Wird geworfen, wenn der
105      *   Index < 0 oder >= Dimension ist.
106      */
107     public void setEntry(int index, double value) {
108         if (index < 0 || index >= _entries.length) throw new IndexOutOfBoundsException();
109         _entries[index] = value;
110     } // public void setEntry(int index, double value) ...
111
112     /**
113      * Gibt eine Kopie des transponierten Vektors zurueck.
114      * @return Kopie des transponierten Vektors.
115      */
116     public IRealRowVector transpose() {
117         return new RealRowVector(_entries);
118     } // public IRealRowVector transpose() ...
119
120     /**
121      * Addiert den angegebenen Wert zum Eintrag mit dem angegebenen Index.
122      * @param index Index des Eintrags.
123      * @param value Wert, der zum Eintrag addiert werden soll.
124      * @exception IndexOutOfBoundsException Wird geworfen, wenn der
125      *   Index < 0 oder >= Dimension ist.
126      */
127     public void addEntry(int index, double value) {
128          if (index < 0 || index >= _entries.length) throw new IndexOutOfBoundsException();
129          _entries[index] += value;
130     } // public void addEntry(int index, double value) ...
131
132     /**
133      * Multipliziert den Eintrag mit dem angegebenen Index mit dem angegebenen Faktor.
134      * @param index Index des Eintrags.
135      * @param factor Faktor, mit dem der Eintrag multipliziert werden soll.
136      * @exception IndexOutOfBoundsException Wird geworfen, wenn der
137      *   Index < 0 oder >= Dimension ist.
138      */
139     public void multiplyEntry (int index, double factor) {
140         if (index < 0 || index >= _entries.length) throw new IndexOutOfBoundsException ();
141         _entries[index] *= factor;
142     } // public void multiplyEntry (int index, double factor) ...
143
144     /**
145      * Addiert den uebergebenen Vektor zum aktuellen Vektor.
146      * Wenn der uebergebene Vektor null ist, wird der Nullvektor addiert.
147      * @param vector Vektor, der addiert werden soll.
148      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Wird geworfen, wenn
           die Dimension
149      *   des uebergebenen Vektors nicht mit der des aktuellen Vektors uebereinstimmt.
150      */
151     public void addVector(IRealColumnVector vector) {
152         if (vector != null) {
153             int dimension = _entries.length;
154             if (vector instanceof RealColumnVector) {
155                 double[] vectorEntries = ((RealColumnVector) vector)._entries;
156                 if (dimension != vectorEntries.length) throw new DimensionException();
157                 for (int c = 0; c < dimension; c++) {
158                     _entries[c] += vectorEntries[c];
```

```
159                 } // for (int c = 0; c < dimension; c++) ...
160             } else { // if (!(vector instanceof RealColumnVector)) ...
161                 if (dimension != vector.getDimension()) throw new DimensionException();
162                 for (int c = 0; c < dimension; c++) {
163                     _entries[c] += vector.getEntry(c);
164                 } // for (int c = 0; c < dimension; c++) ...
165             } // if (!(vector instanceof RealColumnVector)) ...
166         } // if (vector != null) ...
167     } // public void addVector(IRealColumnVector vector) ...
168 
169     /**
170      * Addiert das factor-Fache des uebergebenen Vektors zum aktuellen Vektor.
171      * Wenn der uebergebene Vektor null ist, wird der Nullvektor addiert.
172      * @param vector Vektor, dessen factor-Faches addiert werden soll.
173      * @param factor Faktor, mit dem der uebergebene Vektor multipliziert werden soll.
174      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Wird geworfen, wenn
           die Dimension
175      *   des uebergebenen Vektors nicht mit der des aktuellen Vektors uebereinstimmt.
176      */
177     public void addVector(IRealColumnVector vector, double factor) {
178         if (vector != null) {
179             int dimension = _entries.length;
180             if (vector instanceof RealColumnVector) {
181                 double[] vectorEntries = ((RealColumnVector) vector)._entries;
182                 if (dimension != vectorEntries.length) throw new DimensionException();
183                 for (int c = 0; c < dimension; c++) {
184                     _entries[c] += factor * vectorEntries[c];
185                 } // for (int c = 0; c < dimension; c++) ...
186             } else { // if (!(vector instanceof RealColumnVector)) ...
187                 if (dimension != vector.getDimension()) throw new DimensionException();
188                 for (int c = 0; c < dimension; c++) {
189                     _entries[c] += factor * vector.getEntry(c);
190                 } // for (int c = 0; c < dimension; c++) ...
191             } // if (!(vector instanceof RealColumnVector)) ...
192         } // if (vector != null) ...
193     } // public void addVector(IRealColumnVector vector, double factor) ...
194 
195     /**
196      * Subtrahiert den uebergebenen Vektor vom aktuellen Vektor.
197      * Wenn der uebergebene Vektor null ist, wird der Nullvektor subtrahiert.
198      * @param vector Vektor, der subtrahiert werden soll.
199      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Wird geworfen, wenn
           die Dimension
200      *   des uebergebenen Vektors nicht mit der des aktuellen Vektors uebereinstimmt.
201      */
202     public void subtractVector(IRealColumnVector vector) {
203         if (vector != null) {
204             int dimension = _entries.length;
205             if (vector instanceof RealColumnVector) {
206                 double[] vectorEntries = ((RealColumnVector) vector)._entries;
207                 if (dimension != vectorEntries.length) throw new DimensionException();
208                 for (int c = 0; c < dimension; c++) {
209                     _entries[c] -= vectorEntries[c];
210                 } // for (int c = 0; c < dimension; c++) ...
211             } else { // if (!(vector instanceof RealColumnVector)) ...
212                 if (dimension != vector.getDimension()) throw new DimensionException();
213                 for (int c = 0; c < dimension; c++) {
214                     _entries[c] -= vector.getEntry(c);
215                 } // for (int c = 0; c < dimension; c++) ...
216             } // if (!(vector instanceof RealColumnVector)) ...
217         } // if (vector != null) ...
218     } // public void subtractVector(IRealColumnVector vector) ...
219 
220     /**
221      * Subtrahiert das factor-Fache des uebergebenen Vektors vom aktuellen Vektor.
222      * Wenn der uebergebene Vektor null ist, wird der Nullvektor subtrahiert.
223      * @param vector Vektor, dessen factor-Faches subtrahiert werden soll.
224      * @param factor Faktor, mit dem der uebergebene Vektor multipliziert werden soll.
225      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Wird geworfen, wenn
           die Dimension
226      *   des uebergebenen Vektors nicht mit der des aktuellen Vektors uebereinstimmt.
227      */
228     public void subtractVector(IRealColumnVector vector, double factor) {
229         if (vector != null) {
230             int dimension = _entries.length;
231             if (vector instanceof RealColumnVector) {
232                 double[] vectorEntries = ((RealColumnVector) vector)._entries;
233                 if (dimension != vectorEntries.length) throw new DimensionException();
234                 for (int c = 0; c < dimension; c++) {
235                     _entries[c] -= factor * vectorEntries[c];
```

```
236                 } // for (int c = 0; c < dimension; c++) ...
237             } else { // if (!(vector instanceof RealColumnVector)) ...
238                 if (dimension != vector.getDimension()) throw new DimensionException();
239                 for (int c = 0; c < dimension; c++) {
240                     _entries[c] -= factor * vector.getEntry(c);
241                 } // for (int c = 0; c < dimension; c++) ...
242             } // if (!(vector instanceof RealColumnVector)) ...
243         } // if (vector != null) ...
244     } // public void subtractVector(IRealColumnVector vector, double factor) ...
245 
246     /**
247      * Multipliziert den aktuellen Vektor mit dem uebergebenen Faktor.
248      * @param factor Faktor, mit dem der Vektor multipliziert werden soll.
249      */
250     public void scalarMultiply(double factor) {
251         int dimension = _entries.length;
252         for (int c = 0; c < dimension; c++) {
253             _entries[c] *= factor;
254         } // for (int c = 0; c < dimension; c++) ...
255     } // public void scalarMultiply(double factor) ...
256 
257     /**
258      * Ermittelt die Maximumsnorm des aktuellen Vektors.
259      * @return Maximumsnorm des aktuellen Vektors.
260      */
261     public double getNormMax() {
262         int dimension = _entries.length;
263         double max = 0.0D;
264         double currentElement;
265         // Maximum der Betraege der Koordinaten ermitteln.
266         for (int c = 0; c < dimension; c++) {
267             currentElement = _entries[c];
268             if (currentElement < 0.0D) { currentElement = -currentElement; }
269             if (currentElement > max) { max = currentElement; }
270         } // for (int c = 0; c < dimension; c++) ...
271         return max;
272     } // public double getNormMax() ...
273 
274     /**
275      * Ermittelt die l1-Norm des aktuellen Vektors.
276      * @return l1-Norm des aktuellen Vektors.
277      */
278     public double getNormSum() {
279         int dimension = _entries.length;
280         double sum = 0.0D;
281         double currentElement;
282         // Summe der Betraege der Koordinaten ermitteln.
283         for (int c = 0; c < dimension; c++) {
284             currentElement = _entries[c];
285             sum += currentElement < 0.0D ? -currentElement : currentElement;
286         } // for (int c = 0; c < dimension; c++) ...
287         return sum;
288     } // public double getNormSum() ...
289 
290     /**
291      * Ermittelt die euklidische Norm des aktuellen Vektors.
292      * @return Euklidische Norm des aktuellen Vektors.
293      */
294     public double getNorm() {
295         int dimension = _entries.length;
296         double squareSum = 0.0D;
297         double currentElement;
298         // Summe der Quadrate der Koordinaten ermitteln.
299         for (int c = 0; c < dimension; c++) {
300             currentElement = _entries[c];
301             squareSum += currentElement * currentElement;
302         } // for (int c = 0; c < dimension; c++) ...
303         return Math.sqrt(squareSum);
304     } // public double getNorm() ...
305 
306     /**
307      * Ermittelt das kanonische Skalarprodukt des aktuellen Vektors mit dem
308      * uebergebenen Vektor.
309      * @param vector Vektor, mit dem das kanonische Skalarprodukt gebildet werden soll.
310      *   Wenn vector = null ist, wird mit dem Nullvektor multipliziert.
311      * @return Kanonisches Skalarprodukt des aktuellen Vektors mit dem uebergebenen Vektor.
312      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Dimension des
             uebergebenen Vektors stimmt
313      *   nicht mit der Dimension des aktuellen Vektors ueberein.
314      */
```

```
315     public double getInnerProduct(IRealColumnVector vector) {
316         if (vector != null) {
317             int dimension = _entries.length;
318             if (vector instanceof RealColumnVector) {
319                 double[] vectorEntries = ((RealColumnVector) vector)._entries;
320                 if (dimension != _entries.length) throw new DimensionException();
321                 double result = 0.0D;
322                 for (int c = 0; c < dimension; c++) {
323                     result += _entries[c] * vectorEntries[c];
324                 } // for (int c = 0; c < dimension; c++) ...
325                 return result;
326             } else { // if (!(vector instanceof RealColumnVector)) ...
327                 if (dimension != vector.getDimension()) throw new DimensionException();
328                 double result = 0.0D;
329                 for (int c = 0; c < dimension; c++) {
330                     result += _entries[c] * vector.getEntry(c);
331                 } // for (int c = 0; c < dimension; c++) ...
332                 return result;
333             } // if (!(vector instanceof RealColumnVector)) ...
334         } else { // if (vector == null) ...
335             return 0.0D;
336         } // if (vector == null) ...
337     } // public double getInnerProduct(IRealColumnVector vector) ...
338 } // public class RealColumnVector implements IMutableRealColumnVector ...
```

Listing 22: Klasse *at.techmath.boltzmann.linearalgebra.RealDiagonalMatrix*

```
1  package at.techmath.boltzmann.linearalgebra;
2
3  /**
4   * Repraesentiert eine reellwertige Diagonalmatrix.
5   * @author Stefan Schnabl (e0226245)
6   */
7  public class RealDiagonalMatrix implements IRealDiagonalMatrix {
8      /**
9       * Erzeugt eine Null-Diagonal-Matrix mit der angegebenen Dimension.
10      * @param dimension Dimension der Matrix.
11      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Wird geworfen, wenn
            die
12      *   angegebene Dimension < 1 ist.
13      */
14     public RealDiagonalMatrix(int dimension) {
15         if (dimension < 1) throw new DimensionException("dimension < 1");
16         _entries = new double[dimension];
17     } // public RealDiagonalMatrix(int dimension) ...
18
19     /**
20      * Erzeugt eine Identitaets-Matrix mit der angegebenen Dimension.
21      * @param dimension Dimension der Matrix.
22      * @return Neue Einheitsmatrix mit der angegebenen Dimension.
23      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Wird geworfen, wenn
            die
24      *   angegebene Dimension < 1 ist.
25      */
26     public static RealDiagonalMatrix getUnitMatrix(int dimension) {
27         if (dimension < 1) throw new DimensionException("dimension < 1");
28         RealDiagonalMatrix result = new RealDiagonalMatrix(dimension);
29         double[] resultEntries = result._entries;
30         for (int c = 0; c < dimension; c++) {
31             resultEntries[c] = 1.0D;
32         } // for (int c = 0; c < dimension; c++) ...
33         return result;
34     } // public static RealDiagonalMatrix getUnitMatrix(int dimension) ...
35
36     /**
37      * Gibt eine String-Repraesentation der Matrix zurueck.
38      * @return String-Repraesentation der Matrix.
39      */
40     @Override
41     public String toString() {
42         int dimension = _entries.length;
43         StringBuilder sb = new StringBuilder();
44         sb.append('[');
45         for (int currentRow = 0; currentRow < dimension; currentRow++) {
46             if (currentRow > 0) {
47                 sb.append(',');
48             } // if (currentRow > 0) ...
49             sb.append('(');
50             for (int currentColumn = 0; currentColumn < dimension; currentColumn++) {
51                 if (currentColumn > 0) {
52                     sb.append(',');
53                 } // if (currentColumn > 0) ...
54                 if (currentRow == currentColumn) {
55                     sb.append(_entries[currentRow]);
56                 } else { // if (currentRow != currentColumn) ...
57                     sb.append(0.0D);
58                 } // if (currentRow != currentColumn) ...
59             } // for (int currentColumn = 0; currentColumn < dimension; currentColumn++) ...
60             sb.append(')');
61         } // for (int currentRow = 0; currentRow < dimension; currentRow++) ...
62         sb.append(']');
63         return sb.toString();
64     } // public String toString() ...
65
66     /**
67      * Eintraege der Diagonale.
68      */
69     protected double[] _entries;
70
71     /**
72      * Gibt die Anzahl der Zeilen der Matrix zurueck.
73      * @return Anzahl der Zeilen der Matrix.
74      */
75     public int getRowCount() {
76         return _entries.length;
77     } // public int getRowCount() ...
```

```
78
79      /**
80       * Gibt die Anzahl der Spalten der Matrix zurueck.
81       * @return Anzahl der Spalten der Matrix.
82       */
83      public int getColumnCount() {
84          return _entries.length;
85      } // public int getColumnCount() ...
86
87      /**
88       * Gibt zurueck, ob es sich um eine quadratische Matrix handelt.
89       * @return true, wenn es sich um eine quadratische Matrix handelt, ansonsten false.
90       */
91      public boolean isSquare() {
92          return true;
93      } // public boolean isSquare() ...
94
95      /**
96       * Gibt den Eintrag an der angegebenen Position zurueck.
97       * @param row Zeile des Eintrags (Nullbasiert).
98       * @param column Spalte des Eintrags (Nullbasiert).
99       * @return Eintrag an der angegebenen Position.
100      * @exception IndexOutOfBoundsException Wird geworfen, wenn die angegebene
101      *   Position nicht existiert.
102      */
103     public double getEntry(int row, int column) {
104         int dimension = _entries.length;
105         if (row < 0 || row >= dimension) throw new IndexOutOfBoundsException("row");
106         if (column < 0 || column >= dimension) throw new IndexOutOfBoundsException("column");
107         return row == column ? _entries[row] : 0.0D;
108     } // public double getEntry(int row, int column) ...
109
110     /**
111      * Gibt eine veraenderbare Kopie der aktuellen Matrix zurueck.
112      * @return Veraenderbare Kopie der aktuellen Matrix.
113      */
114     public IMutableRealMatrix getMutableCopy() {
115         int dimension = _entries.length;
116         RealMatrix result = new RealMatrix(dimension, dimension);
117         for (int c = 0; c < dimension; c++) {
118             result._entries[c * dimension + c] = _entries[c];
119         } // for (int c = 0; c < dimension; c++) ...
120         return result;
121     } // public IMutableRealMatrix getMutableCopy() ...
122
123     /**
124      * Multipliziert die Matrix von links mit dem angegebenen Zeilenvektor
125      * und gibt das Ergebnis als veraenderbaren Vektor zurueck.
126      * Die aktuelle Matrix bleibt unangetastet.
127      * @param vector Vektor, mit dem die Matrix multipliziert werden soll.
128      *   (null wird als Nullvektor interpraetiert)
129      * @return Ergebnisvektor.
130      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Wird geworfen, wenn
            die Dimensionen
131      *   nicht uebereinstimmen.
132      */
133     public IMutableRealRowVector leftMultiplyRowVector(IRealRowVector vector) {
134         if (vector == null) {
135             // Der Vektor wird als Nullvektor mit der Dimension der Matrix interpraetiert.
136             return new RealRowVector(_entries.length);
137         } else { // if (vector != null) ...
138             if (vector instanceof RealRowVector) {
139                 // RealRowVector-Instanz
140                 double[] vectorEntries = ((RealRowVector) vector)._entries;
141                 double[] matrixEntries = _entries;
142                 int dimension = matrixEntries.length;
143                 if (vectorEntries.length != dimension) throw new DimensionException();
144                 RealRowVector result = new RealRowVector(dimension);
145                 double[] resultEntries = result._entries;
146                 for (int currentColumn = 0; currentColumn < dimension; currentColumn++) {
147                     resultEntries[currentColumn] = vectorEntries[currentColumn] * matrixEntries
                            [currentColumn];
148                 } // for (int currentColumn = 0; currentColumn < dimension; currentColumn++)
                        ...
149                 return result;
150             } else { // if (!(vector instanceof RealRowVector)) ...
151                 // Allgemeiner Vektor
152                 double[] matrixEntries = _entries;
153                 int dimension = matrixEntries.length;
154                 if (vector.getDimension() != dimension) throw new DimensionException();
```

```
155                 RealRowVector result = new RealRowVector(dimension);
156                 double[] resultEntries = result._entries;
157                 for (int currentColumn = 0; currentColumn < dimension; currentColumn++) {
158                     resultEntries[currentColumn] = vector.getEntry(currentColumn) *
                            matrixEntries[currentColumn];
159                 } // for (int currentColumn = 0; currentColumn < dimension; currentColumn++)
                        ...
160                 return result;
161             } // if (!(vector instanceof RealRowVector)) ...
162         } // if (vector != null) ...
163     } // public IMutableRealRowVector leftMultiplyRowVector(IRealRowVector vector) ...
164
165     /**
166      * Multipliziert die Matrix von rechts mit dem angegebenen Zeilenvektor
167      * und gibt das Ergebnis als Matrix zurueck. Die aktuelle Matrix bleibt
168      * unangetastet.
169      * @param vector Vektor, mit dem die Matrix multipliziert werden soll.
170      * @return Ergebnismatrix.
171      * @exception IllegalArgumentException Wird geworfen, wenn der angegebene Vektor null ist.
172      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Wird geworfen, wenn
173      *   die Dimensionen nicht uebereinstimmen.
174      */
175     public IRealMatrix rightMultiplyRowVector(IRealRowVector vector) {
176         if (vector == null) throw new IllegalArgumentException("vector = null.");
177         double[] diagonalEntries = _entries;
178         if (diagonalEntries.length != 1) throw new DimensionException();
179         if (vector instanceof RealRowVector) {
180             double[] vectorEntries = ((RealRowVector) vector)._entries;
181             int columnCount = vectorEntries.length;
182             double matrixEntry = diagonalEntries[0];
183             RealMatrix result = new RealMatrix(1, columnCount);
184             double[] resultEntries = result._entries;
185             for (int currentColumn = 0; currentColumn < columnCount; currentColumn++) {
186                 resultEntries[currentColumn] = matrixEntry * vectorEntries[currentColumn];
187             } // for (int currentColumn = 0; currentColumn < columnCount; currentColumn++) ...
188             return result;
189         } else { // if (!(vector instanceof RealRowVector)) ...
190             int columnCount = vector.getDimension();
191             double matrixEntry = diagonalEntries[0];
192             RealMatrix result = new RealMatrix(1, columnCount);
193             double[] resultEntries = result._entries;
194             for (int currentColumn = 0; currentColumn < columnCount; currentColumn++) {
195                 resultEntries[currentColumn] = matrixEntry * vector.getEntry(currentColumn);
196             } // for (int currentColumn = 0; currentColumn < columnCount; currentColumn++) ...
197             return result;
198         } // if (!(vector instanceof RealRowVector)) ...
199     } // public IRealMatrix rightMultiplyRowVector(IRealRowVector vector) ...
200
201     /**
202      * Multipliziert die Matrix von links mit dem angegebenen Spaltenvektor
203      * und gibt das Ergebnis als Matrix zurueck. Die aktuelle Matrix bleibt
204      * unangetastet.
205      * @param vector Vektor, mit dem die Matrix multipliziert werden soll.
206      * @return Ergebnismatrix.
207      * @exception IllegalArgumentException Wird geworfen, wenn der angegebene Vektor null ist.
208      * @exception at.techmath.boltzmann.linearalgebra.DimensionExceptino Wird geworfen, wenn
209      *   die Dimensionen nicht uebereinstimmen.
210      */
211     public IRealMatrix leftMultiplyColumnVector(IRealColumnVector vector) {
212         if (vector == null) throw new IllegalArgumentException("vector = null.");
213         double[] diagonalEntries = _entries;
214         if (diagonalEntries.length != 1) throw new DimensionException();
215         if (vector instanceof RealRowVector) {
216             double[] vectorEntries = ((RealRowVector) vector)._entries;
217             int rowCount = vectorEntries.length;
218             double matrixEntry = diagonalEntries[0];
219             RealMatrix result = new RealMatrix(rowCount, 1);
220             double[] resultEntries = result._entries;
221             for (int currentRow = 0; currentRow < rowCount; currentRow++) {
222                 resultEntries[currentRow] = matrixEntry * vectorEntries[currentRow];
223             } // for (int currentRow = 0; currentRow < rowCount; currentRow++) ...
224             return result;
225         } else { // if (!(vector instanceof RealRowVector)) ...
226             int rowCount = vector.getDimension();
227             double matrixEntry = diagonalEntries[0];
228             RealMatrix result = new RealMatrix(rowCount, 1);
229             double[] resultEntries = result._entries;
230             for (int currentRow = 0; currentRow < rowCount; currentRow++) {
231                 resultEntries[currentRow] = matrixEntry * vector.getEntry(currentRow);
232             } // for (int currentRow = 0; currentRow < rowCount; currentRow++) ...
```

```
233             return result;
234         } // if (!(vector instanceof RealRowVector)) ...
235     } // public IRealMatrix leftMultiplyColumnVector(IRealColumnVector vector) ...
236 
237     /**
238      * Multipliziert die Matrix von rechts mit dem angegebenen Spaltenvektor
239      * und gibt das Ergebnis als veraenderbaren Vektor zurueck.
240      * Die aktuelle Matrix bleibt unangetastet.
241      * @param vector Vektor, mit dem die Matrix multipliziert werden soll.
242      *   (null wird als Nullvektor interpraetiert)
243      * @return Ergebnisvektor.
244      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Wird geworfen, wenn
             die Dimensionen
245      *   nicht uebereinstimmen.
246      */
247     public IMutableRealColumnVector rightMultiplyColumnVector(IRealColumnVector vector) {
248         if (vector == null) {
249             // Uebergebener Vektor wird als Nullvektor mit der selben Dimension wie die der
                  Matrix interpraetiert.
250             return new RealColumnVector(_entries.length);
251         } else { // if (vector != null) ...
252             if (vector instanceof RealColumnVector) {
253                 // RealColumnVector-Instanz
254                 double[] diagonalEntries = _entries;
255                 int dimension = diagonalEntries.length;
256                 double[] vectorEntries = ((RealColumnVector) vector)._entries;
257                 if (vectorEntries.length != dimension) throw new DimensionException();
258                 RealColumnVector result = new RealColumnVector(dimension);
259                 double[] resultEntries = result._entries;
260                 for (int currentRow = 0; currentRow < dimension; currentRow++) {
261                     resultEntries[currentRow] = diagonalEntries[currentRow] * vectorEntries[
                          currentRow];
262                 } // for (int currentRow = 0; currentRow < dimension; currentRow++) ...
263                 return result;
264             } else { // if (!(vector instanceof RealColumnVector)) ...
265                 // Allgemeiner Vektor
266                 double[] diagonalEntries = _entries;
267                 int dimension = diagonalEntries.length;
268                 if (vector.getDimension() != dimension) throw new DimensionException();
269                 RealColumnVector result = new RealColumnVector(dimension);
270                 double[] resultEntries = result._entries;
271                 for (int currentRow = 0; currentRow < dimension; currentRow++) {
272                     resultEntries[currentRow] = diagonalEntries[currentRow] * vector.getEntry(
                          currentRow);
273                 } // for (int currentRow = 0; currentRow < dimension; currentRow++) ...
274                 return result;
275             } // if  (!(vector instanceof RealColumnVector)) ...
276         } // if (vector != null) ...
277     } // public IMutableRealColumnVector rightMultiplyColumnVector(IRealColumnVector vector)
          ...
278 
279     /**
280      * Multipliziert die Matrix von links mit der angegebenen Matrix
281      * und gibt das Ergebnis als Matrix zurueck. Die aktuelle Matrix bleibt
282      * unangetastet.
283      * @param matrix Matrix, mit der multipliziert werden soll.
284      * @return Ergebnismatrix.
285      * @exception IllegalArgumentException Wird geworfen, wenn die uebergebene Matrix null ist.
286      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Wird geworfen, wenn
287      *   die Dimensionen nicht uebereinstimmen.
288      */
289     public IRealMatrix leftMultiplyMatrix(IRealMatrix matrix) {
290         if (matrix == null) throw new IllegalArgumentException("matrix = null.");
291         if (matrix instanceof RealMatrix) {
292             RealMatrix leftMatrix = (RealMatrix) matrix;
293             double[] leftEntries = leftMatrix._entries;
294             double[] rightEntries = _entries;
295             int rowCount = leftMatrix._rowCount;
296             int columnCount = rightEntries.length;
297             if (leftMatrix._columnCount != columnCount) throw new DimensionException();
298             RealMatrix result = new RealMatrix(rowCount, columnCount);
299             double[] resultEntries = result._entries;
300             int currentColumn = 0, currentIndex = 0, maxIndex = resultEntries.length;
301             while (currentIndex < maxIndex) {
302                 resultEntries[currentIndex] = leftEntries[currentIndex] * rightEntries[
                      currentColumn];
303                 currentIndex++;
304                 currentColumn++;
305                 if (currentColumn >= columnCount) {
306                     currentColumn = 0;
```

```
307                 } // if (currentColumn >= columnCount) ...
308             } // while (currentIndex < maxIndex) ...
309             return result;
310         } else if (matrix instanceof RealDiagonalMatrix) {
311             double[] leftEntries = ((RealDiagonalMatrix) matrix)._entries;
312             double[] rightEntries = _entries;
313             int dimension = rightEntries.length;
314             if (leftEntries.length != dimension) throw new DimensionException();
315             RealDiagonalMatrix result = new RealDiagonalMatrix(dimension);
316             double[] resultEntries = result._entries;
317             for (int currentColumn = 0; currentColumn < dimension; currentColumn++) {
318                 resultEntries[currentColumn] = leftEntries[currentColumn] * rightEntries[
                        currentColumn];
319             } // for (int currentColumn = 0; currentColumn < dimension; currentColumn++) ...
320             return result;
321         } else if (matrix instanceof RealPermutationMatrix) {
322             // Sei D = (d_ij) = (di delta_i_j) eine Diagonalmatrix,
323             // P = (p_ij) = (delta_i_sigma(j)) eine Permutationsmatrix zur Permutation sigma
                    aus Sn
324             // und C = (c_ij) = P * D, dann gilt:
325             // c_ij = sum_{k=1..n} p_ik * d_kj = sum_{k=1..n} dk * delta_i_sigma(k) * delta_k_j
                    =
326             // = sum_{k=j} dk * delta_i_sigma(k) = dj * delta_i_sigma(j) =
327             // = { dj falls i = sigma(j), 0 sonst }
328             // also steht in der j-ten Spalte der Ergebnis-Matrix der j-te Diagonaleintrag in
                    der sigma(j)-ten Zeile.
329             double[] rightDiagonal = _entries;
330             int[] permutationLeft = ((RealPermutationMatrix) matrix)._permutation;
331             int dimension = _entries.length;
332             if (permutationLeft.length != dimension) throw new DimensionException();
333             RealMatrix result = new RealMatrix(dimension, dimension);
334             double[] resultEntries = result._entries;
335             for (int currentColumn = 0; currentColumn < dimension; currentColumn++) {
336                 resultEntries[permutationLeft[currentColumn] * dimension + currentColumn] =
                        rightDiagonal[currentColumn];
337             } // for (int currentColumn = 0; currentColumn < dimension; currentColumn++) ...
338             return result;
339         } else { // Allgemeiner Fall ...
340             double[] rightEntries = _entries;
341             int rowCount = matrix.getRowCount();
342             int columnCount = rightEntries.length;
343             if (matrix.getColumnCount() != columnCount) throw new DimensionException();
344             RealMatrix result = new RealMatrix(rowCount, columnCount);
345             double[] resultEntries = result._entries;
346             int currentColumn = 0, currentRow = 0, currentIndex = 0, maxIndex = resultEntries.
                    length;
347             while (currentIndex < maxIndex) {
348                 resultEntries[currentIndex] = matrix.getEntry(currentRow, currentColumn) *
                        rightEntries[currentColumn];
349                 currentIndex++;
350                 currentColumn++;
351                 if (currentColumn >= columnCount) {
352                     currentColumn = 0;
353                     currentRow++;
354                 } // if (currentColumn >= columnCount) ...
355             } // while (currentIndex < maxIndex) ...
356             return result;
357         } // Allgemeiner Fall ...
358     } // public IRealMatrix leftMultiplyMatrix(IRealMatrix matrix) ...
359
360     /**
361      * Multipliziert die Matrix von rechts mit der angegebenen Matrix
362      * und gibt das Ergebnis als Matrix zurueck. Die aktuelle Matrix bleibt
363      * unangetastet.
364      * @param matrix Matrix, mit der multipliziert werden soll.
365      * @return Ergebnismatrix.
366      * @exception IllegalArgumentException Wird geworfen, wenn die uebergebene Matrix null ist.
367      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Wird geworfen, wenn
368      *   die Dimensionen nicht uebereinstimmen.
369      */
370     public IRealMatrix rightMultiplyMatrix(IRealMatrix matrix) {
371         if (matrix == null) throw new IllegalArgumentException("matrix = null.");
372         if (matrix instanceof RealMatrix) {
373             RealMatrix rightMatrix = (RealMatrix) matrix;
374             double[] leftEntries = _entries;
375             double[] rightEntries = rightMatrix._entries;
376             int rowCount = leftEntries.length;
377             int columnCount = rightMatrix._columnCount;
378             if (rowCount != rightMatrix._rowCount) throw new DimensionException();
379             RealMatrix result = new RealMatrix(rowCount, columnCount);
```

```
380             double[] resultEntries = result._entries;
381             int currentRow = 0, currentColumn = 0, currentIndex = 0, maxIndex = resultEntries.
                    length;
382             double currentFactor = leftEntries[currentRow];
383             while (currentIndex < maxIndex) {
384                 resultEntries[currentIndex] = currentFactor * rightEntries[currentIndex];
385                 currentIndex++;
386                 currentColumn++;
387                 if (currentColumn >= columnCount) {
388                     currentColumn = 0;
389                     currentRow++;
390                     if (currentRow < rowCount) {
391                         currentFactor = leftEntries[currentRow];
392                     } // if (currentRow < rowCount) ...
393                 } // if (currentColumn >= columnCount) ...
394             } // while (currentIndex < maxIndex) ...
395             return result;
396         } else if (matrix instanceof RealDiagonalMatrix) {
397             double[] leftEntries = _entries;
398             double[] rightEntries = ((RealDiagonalMatrix) matrix)._entries;
399             int dimension = leftEntries.length;
400             if (dimension != rightEntries.length) throw new DimensionException();
401             RealDiagonalMatrix result = new RealDiagonalMatrix(dimension);
402             double[] resultEntries = result._entries;
403             for (int currentColumn = 0; currentColumn < dimension; currentColumn++) {
404                 resultEntries[currentColumn] = leftEntries[currentColumn] * rightEntries[
                        currentColumn];
405             } // for (int currentColumn = 0; currentColumn < dimension; currentColumn++) ...
406             return result;
407         } else if (matrix instanceof RealPermutationMatrix) {
408             // Sei D = (d_ij) = (dj delta_i_j) eine Diagonalmatrix,
409             // P = (p_ij) = (delta_i_sigma(j)) eine Permutationsmatrix zur Permutation sigma
                    aus Sn
410             // und C = (c_ij) = D * P, dann gilt:
411             // c_ij = sum_{k=1..n} d_ik * p_kj = sum_{k=1..n} dk * delta_i_k * delta_k_sigma(j)
                        =
412             // = di * delta_i_sigma(j) =
413             // = { di falls i = sigma(j), 0 sonst } =
414             // = { di falls j = sigma^(-1)(i), 0 sonst }
415             // also steht in der i-ten Zeile der Ergebnis-Matrix der i-te Diagonaleintrag in
                    der sigma^(-1)(i)-ten Spalte.
416             double[] leftDiagonal = _entries;
417             int[] permutationInverseRight = ((RealPermutationMatrix) matrix).
                    _permutationInverse;
418             int dimension = leftDiagonal.length;
419             if (permutationInverseRight.length != dimension) throw new DimensionException();
420             RealMatrix result = new RealMatrix(dimension, dimension);
421             double[] resultEntries = result._entries;
422             for (int currentRow = 0; currentRow < dimension; currentRow++) {
423                 resultEntries[currentRow * dimension + permutationInverseRight[currentRow]] =
                        leftDiagonal[currentRow];
424             } // for (int currentRow = 0; currentRow < dimension; currentRow++) ...
425             return result;
426         } else { // Allgemeiner Fall ...
427             double[] leftEntries = _entries;
428             int rowCount = leftEntries.length;
429             int columnCount = matrix.getColumnCount();
430             if (rowCount != matrix.getRowCount()) throw new DimensionException();
431             RealMatrix result = new RealMatrix(rowCount, columnCount);
432             double[] resultEntries = result._entries;
433             int currentRow = 0, currentColumn = 0, currentIndex = 0, maxIndex = resultEntries.
                    length;
434             double currentFactor = leftEntries[currentRow];
435             while (currentIndex < maxIndex) {
436                 resultEntries[currentIndex] = currentFactor * matrix.getEntry(currentRow,
                        currentColumn);
437                 currentIndex++;
438                 currentColumn++;
439                 if (currentColumn >= columnCount) {
440                     currentColumn = 0;
441                     currentRow++;
442                     if (currentRow < rowCount) {
443                         currentFactor = leftEntries[currentRow];
444                     } // if (currentRow < rowCount) ...
445                 } // if (currentColumn >= columnCount) ...
446             } // while (currentIndex < maxIndex) ...
447             return result;
448         } // Allgemeiner Fall ...
449     } // public IRealMatrix rightMultiplyMatrix(IRealMatrix matrix) ...
450
```

```
451     /**
452      * Gibt die Dimension der Matrix zurueck.
453      * @return Dimension der Matrix.
454      */
455     public int getDimension() {
456         return _entries.length;
457     } // public int getDimension() ...
458 
459     /**
460      * Gibt den Diagonaleintrag mit dem angegebenen Index zurueck.
461      * @param index Gewuenschter Index.
462      * @return Diagonaleintrag mit dem angegebenen Index (Nullbasiert).
463      * @exception IndexOutOfBoundsException Wird geworfen, wenn der uebergebene
464      *   Index nicht existiert.
465      */
466     public double getDiagonalEntry(int index) {
467         if (index < 0 || index >= _entries.length) throw new IndexOutOfBoundsException();
468         return _entries[index];
469     } // public double getDiagonalEntry(int index) ...
470 
471     /**
472      * Setzt den Diagonaleintrag mit dem angegebenen Index
473      * auf den angegebenen Wert.
474      * @param index Index des Eintrags, der gesetzt werden soll (Nullbasiert).
475      * @param value Wert, auf den der Eintrag gesetzt werden soll.
476      * @exception IndexOutOfBoundsException Wird geworfen, wenn der uebergebene
477      *   Index nicht existiert.
478      */
479     public void setDiagonalEntry(int index, double value) {
480         if (index < 0 || index >= _entries.length) throw new IndexOutOfBoundsException();
481         _entries[index] = value;
482     } // public void setDiagonalEntry(int index, double value) ...
483 
484     /**
485      * Addiert zum Diagonaleintrag mit dem angegebenen Index
486      * den angegebenen Wert.
487      * @param index Index des Eintrags, zu dem der Wert addiert werden soll (Nullbasiert).
488      * @param value Wert, der addiert werden soll.
489      * @exception IndexOutOfBoundsException Wird geworfen, wenn der uebergebene
490      *   Index nicht existiert.
491      */
492     public void addDiagonalEntry(int index, double value) {
493         if (index < 0 || index >= _entries.length) throw new IndexOutOfBoundsException();
494         _entries[index] += value;
495     } // public void addDiagonalEntry(int index, double value) ...
496 
497     /**
498      * Multipliziert den Diagonaleintrag mit dem angegebenen Index
499      * mit dem angegebenen Faktor.
500      * @param index Index des Eintrags, der mit dem Faktor multipliziert werden soll (
           Nullbasiert).
501      * @param factor Faktor, mit dem multipliziert werden soll.
502      * @exception IndexOutOfBoundsException Wird geworfen, wenn der uebergebene
503      *   Index nicht existiert.
504      */
505     public void multiplyDiagonalEntry(int index, double factor) {
506         if (index < 0 || index >= _entries.length) throw new IndexOutOfBoundsException();
507         _entries[index] *= factor;
508     } // public void multiplyDiagonalEntry(int index, double factor) ...
509 } // public class RealDiagonalMatrix implements IRealDiagonalMatrix ...
```

Listing 23: Klasse *at.techmath.boltzmann.linearalgebra.RealMatrix*

```
1  package at.techmath.boltzmann.linearalgebra;
2
3  /**
4   * Repraesentiert eine reellwertige Matrix, bei der jeder Eintrag veraendert werden kann.
5   * @author Stefan Schnabl (e0226245)
6   */
7  public class RealMatrix implements IMutableRealMatrix {
8      /**
9       * Erzeugt eine neue RealMatrix-Instanz mit der
10      * angegebenen Dimension, die auf 0 initialisiert ist.
11      * @param rowCount Anzahl der Zeilen.
12      * @param columnCount Anzahl der Spalten.
13      * @exception IllegalArgumentException Wird geworfen, wenn
14      *   die Zeilenanzahl oder die Spaltenanzahl <= 0 ist.
15      */
16     public RealMatrix(int rowCount, int columnCount) {
17         if (rowCount <= 0) throw new IllegalArgumentException("rowCount <= 0");
18         if (columnCount <= 0) throw new IllegalArgumentException("columnCount <= 0");
19         _rowCount = rowCount;
20         _columnCount = columnCount;
21         _entries = new double[rowCount * columnCount];
22     } // public RealMatrix (int rowCount, int columnCount) ...
23
24     /**
25      * Erzeugt eine Kopie der uebergebenen Matrix.
26      * @param matrix Matrix, die kopiert werden soll.
27      * @exception IllegalArgumentException Wird geworfen, wenn
28      *   die uebergebene Matrix null ist.
29      */
30     public RealMatrix(IRealMatrix matrix) {
31         if (matrix == null) throw new IllegalArgumentException();
32         if (matrix instanceof RealMatrix) {
33             // RealMatrix-Instanz
34             RealMatrix realMatrix = (RealMatrix) matrix;
35             _rowCount = realMatrix._rowCount;
36             _columnCount = realMatrix._columnCount;
37             double[] realMatrixEntries = realMatrix._entries;
38             _entries = new double[realMatrixEntries.length];
39             for (int c = 0; c < _entries.length; c++) {
40                 _entries[c] = realMatrixEntries[c];
41             } // for (int c = 0; c < _entries.length; c++) ...
42         } else { // if (!(matrix instanceof RealMatrix)) ...
43             // Allgemeine Matrix
44             int rowCount = _rowCount = matrix.getRowCount();
45             int columnCount = _columnCount = matrix.getColumnCount();
46             int length = rowCount * columnCount;
47             _entries = new double[length];
48             int currentRow = 0;
49             int currentColumn = 0;
50             for (int c = 0; c < length; c++) {
51                 _entries[c] = matrix.getEntry(currentRow, currentColumn);
52                 currentColumn++;
53                 if (currentColumn >= columnCount) {
54                     currentColumn = 0;
55                     currentRow++;
56                 } // if (currentColumn >= columnCount) ...
57             } // for (int c = 0; c < length; c++) ...
58         } // if (!(matrix instanceof RealMatrix)) ...
59     } // public RealMatrix (IRealMatrix matrix) ...
60
61     /**
62      * Gibt die Einheitsmatrix mit der angegebenen Dimension zurueck.
63      * @param dimension Gewuenschte Dimension.
64      * @return Einheitsmatrix mit der angegebenen Dimension.
65      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Wird geworfen, wenn
66      *   die angegebene Dimension < 1 ist.
67      */
68     public static RealMatrix getUnitMatrix(int dimension) {
69         if (dimension < 1) throw new DimensionException();
70         RealMatrix result = new RealMatrix(dimension, dimension);
71         double[] resultEntries = result._entries;
72         for (int currentRow = 0; currentRow < dimension; currentRow++) {
73             resultEntries[currentRow * dimension + currentRow] = 1.0D;
74         } // for (int currentRow = 0; currentRow < dimension; currentRow++) ...
75         return result;
76     } // public static RealMatrix getUnitMatrix(int dimension) ...
77
78     /**
79      * Gibt eine String-Repraesentation der Matrix zurueck.
```

```
80       * @return String-Repraesentation der Matrix.
81       */
82      @Override
83      public String toString() {
84          StringBuilder sb = new StringBuilder();
85          sb.append('[');
86          for (int currentRow = 0; currentRow < _rowCount; currentRow++) {
87              if (currentRow > 0) {
88                  sb.append(',');
89              } // if (currentRow > 0) ...
90              sb.append('(');
91              for (int currentColumn = 0; currentColumn < _columnCount; currentColumn++) {
92                  if (currentColumn > 0) {
93                      sb.append(',');
94                  } // if (currentColumn > 0) ...
95                  sb.append(_entries[currentRow * _columnCount + currentColumn]);
96              } // for (int currentColumn = 0; currentColumn < _columnCount; currentColumn++) ...
97              sb.append(')');
98          } // for (int currentRow = 0; currentRow < _rowCount; currentRow++) ...
99          sb.append(']');
100         return sb.toString();
101     } // public String toString() ...
102
103     /**
104      * Eintraege der Matrix. Dimension des Arrays ist rowCount * columnCount.
105      * Der Eintrag [i, j] der Matrix befindet sich am Index i * columnCount + j.
106      */
107     protected double[] _entries;
108
109     /**
110      * Anzahl der Zeilen (>= 1).
111      */
112     protected int _rowCount;
113
114     /**
115      * Gibt die Anzahl der Zeilen der Matrix zurueck.
116      * @return Anzahl der Zeilen der Matrix.
117      */
118     public int getRowCount() {
119         return _rowCount;
120     } // public int getRowCount() ...
121
122     /**
123      * Anzahl der Spalten (>= 1).
124      */
125     protected int _columnCount;
126
127     /**
128      * Gibt die Anzahl der Spalten der Matrix zurueck.
129      * @return Anzahl der Spalten der Matrix.
130      */
131     public int getColumnCount() {
132         return _columnCount;
133     } // public int getColumnCount() ...
134
135     /**
136      * Gibt zurueck, ob es sich um eine quadratische Matrix handelt.
137      * @return true, wenn es sich um eine quadratische Matrix handelt, ansonsten false.
138      */
139     public boolean isSquare() {
140         return _rowCount == _columnCount;
141     } // public boolean isSquare() ...
142
143     /**
144      * Gibt den Eintrag an der angegebenen Position zurueck.
145      * @param row Zeile des Eintrags (Nullbasiert).
146      * @param column Spalte des Eintrags (Nullbasiert).
147      * @return Eintrag an der angegebenen Position.
148      * @exception IndexOutOfBoundsException Wird geworfen, wenn die angegebene
149      *   Position nicht existiert.
150      */
151     public double getEntry(int row, int column) {
152         if (row < 0 || row >= _rowCount) throw new IndexOutOfBoundsException("row");
153         int columnCount = _columnCount;
154         if (column < 0 || column >= columnCount) throw new IndexOutOfBoundsException("column");
155         return _entries[row * columnCount + column];
156     } // public double getEntry(int row, int column) ...
157
158     /**
159      * Gibt eine veraenderbare Kopie der aktuellen Matrix zurueck.
```

```
160      * @return Veraenderbare Kopie der aktuellen Matrix.
161      */
162     public IMutableRealMatrix getMutableCopy() {
163         return new RealMatrix(this);
164     } // public IMutableRealMatrix getMutableCopy() ...
165 
166     /**
167      * Multipliziert die Matrix von links mit dem angegebenen Zeilenvektor
168      * und gibt das Ergebnis als veraenderbaren Vektor zurueck.
169      * Die aktuelle Matrix bleibt unangetastet.
170      * @param vector Vektor, mit dem die Matrix multipliziert werden soll.
171      *   (null wird als Nullvektor interpraetiert)
172      * @return Ergebnisvektor.
173      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Wird geworfen, wenn
         die Dimensionen
174      *   nicht uebereinstimmen.
175      */
176     public IMutableRealRowVector leftMultiplyRowVector(IRealRowVector vector) {
177         if (vector == null) {
178             return new RealRowVector(_columnCount);
179         } else { // if (vector != null) ...
180             if (vector instanceof RealRowVector) {
181                 // Instanz von RealRowVector
182                 double[] vectorEntries = ((RealRowVector) vector)._entries;
183                 int rowCount = _rowCount;
184                 if (rowCount != vectorEntries.length) throw new DimensionException();
185                 int columnCount = _columnCount;
186                 int maxIndex = _entries.length;
187                 RealRowVector result = new RealRowVector(columnCount);
188                 double[] resultEntries = result._entries;
189                 int currentIndex = 0, currentColumn = 0, currentRow = 0;
190                 // Linearkombination der Zeilenvektoren der Matrix bilden,
191                 // die Koeffizienten sind die Spalteneintraege des Vektors.
192                 double currentFactor = vectorEntries[currentRow];
193                 while (currentIndex < maxIndex) {
194                     resultEntries[currentColumn] += currentFactor * _entries[currentIndex];
195                     currentIndex++;
196                     currentColumn++;
197                     if (currentColumn >= columnCount) {
198                         currentColumn = 0;
199                         currentRow++;
200                         if (currentRow < rowCount) {
201                             currentFactor = vectorEntries[currentRow];
202                         } // if (currentRow < rowCount) ...
203                     } // if (currentColumn >= columnCount) ...
204                 } // while (currentIndex < maxIndex) ...
205                 return result;
206             } else { // if (!(vector instanceof RealRowVector) ...
207                 // Allgemeine IRowVector-Instanz
208                 int rowCount = _rowCount;
209                 if (rowCount != vector.getDimension()) throw new DimensionException();
210                 int columnCount = _columnCount;
211                 int maxIndex = _entries.length;
212                 RealRowVector result = new RealRowVector(columnCount);
213                 double[] resultEntries = result._entries;
214                 int currentIndex = 0, currentColumn = 0, currentRow = 0;
215                 // Linearkombination der Zeilenvektoren der Matrix bilden,
216                 // die Koeffizienten sind die Spalteneintraege des Vektors.
217                 double currentFactor = vector.getEntry(currentRow);
218                 while (currentIndex < maxIndex) {
219                     resultEntries[currentColumn] += currentFactor * _entries[currentIndex];
220                     currentIndex++;
221                     currentColumn++;
222                     if (currentColumn >= columnCount) {
223                         currentColumn = 0;
224                         currentRow++;
225                         if (currentRow < rowCount) {
226                             currentFactor = vector.getEntry(currentRow);
227                         } // if (currentRow < rowCount) ...
228                     } // if (currentColumn >= columnCount) ...
229                 } // while (currentIndex < maxIndex) ...
230                 return result;
231             } // if (!(vector instanceof RealRowVector) ...
232         } // if (vector != null) ...
233     } // public IMutableRealRowVector leftMultiplyRowVector(IRealRowVector vector) ...
234 
235     /**
236      * Multipliziert die Matrix von rechts mit dem angegebenen Zeilenvektor
237      * und gibt das Ergebnis als Matrix zurueck. Die aktuelle Matrix bleibt
238      * unangetastet.
```

```
239      * @param vector Vektor, mit dem die Matrix multipliziert werden soll.
240      * @return Ergebnismatrix.
241      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Wird geworfen, wenn
             die Dimensionen
242      *   nicht uebereinstimmen oder der angegebene Vektor null ist.
243      */
244     public IRealMatrix rightMultiplyRowVector(IRealRowVector vector) {
245         if (vector == null) throw new IllegalArgumentException("vector = null.");
246         if (_columnCount != 1) throw new DimensionException();
247         if (vector instanceof RealRowVector) {
248             // Vektor ist eine RealRowVector-Instanz
249             double[] vectorEntries = ((RealRowVector) vector)._entries;
250             double[] matrixEntries = this._entries; // Die Matrix hat nur eine Spalte -->
                    Eigentlich Spaltenvektor, jeder Index ist eine Zeile.
251             // Die neue Matrix hat _rowCount Zeilen und vectorEntries.length Spalten
252             int rowCount = _rowCount;
253             int columnCount = vectorEntries.length;
254             RealMatrix result = new RealMatrix(rowCount, columnCount);
255             double[] resultEntries = result._entries;
256             int maxIndex = resultEntries.length;
257             int currentRow = 0, currentColumn = 0, currentIndex = 0;
258             double currentFactor = matrixEntries[currentRow];
259             while (currentIndex < maxIndex) {
260                 resultEntries[currentIndex] = currentFactor * vectorEntries[currentColumn];
261                 currentColumn++;
262                 currentIndex++;
263                 if (currentColumn >= columnCount) {
264                     currentColumn = 0;
265                     currentRow++;
266                     if (currentRow < rowCount) {
267                         currentFactor = matrixEntries[currentRow];
268                     } // if (currentRow < rowCount) ...
269                 } // if (currentColumn >= columnCount) ...
270             } // while (currentIndex < maxIndex) ...
271             return result;
272         } else { // if (!(vector instanceof RealRowVector)) ...
273             // Allgemeiner Vektor
274             double[] matrixEntries = this._entries; // Die Matrix hat nur eine Spalte -->
                    Eigentlich Spaltenvektor, jeder Index ist eine Zeile.
275             // Die neue Matrix hat _rowCount Zeilen und vector.getDimension() Spalten
276             int rowCount = _rowCount;
277             int columnCount = vector.getDimension();
278             RealMatrix result = new RealMatrix(rowCount, columnCount);
279             double[] resultEntries = result._entries;
280             int maxIndex = resultEntries.length;
281             int currentRow = 0, currentColumn = 0, currentIndex = 0;
282             double currentFactor = matrixEntries[currentRow];
283             while (currentIndex < maxIndex) {
284                 resultEntries[currentIndex] = currentFactor * vector.getEntry(currentColumn);
285                 currentColumn++;
286                 currentIndex++;
287                 if (currentColumn >= columnCount) {
288                     currentColumn = 0;
289                     currentRow++;
290                     if (currentRow < rowCount) {
291                         currentFactor = matrixEntries[currentRow];
292                     } // if (currentRow < rowCount) ...
293                 } // if (currentColumn >= columnCount) ...
294             } // while (currentIndex < maxIndex) ...
295             return result;
296         } // if (!(vector instanceof RealRowVector)) ...
297     } // public IRealMatrix rightMultiplyRowVector(IRealRowVector vector) ...
298
299     /**
300      * Multipliziert die Matrix von links mit dem angegebenen Spaltenvektor
301      * und gibt das Ergebnis als Matrix zurueck. Die aktuelle Matrix bleibt
302      * unangetastet.
303      * @param vector Vektor, mit dem die Matrix multipliziert werden soll.
304      * @return Ergebnismatrix.
305      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Wird geworfen, wenn
             die Dimensionen
306      *   nicht uebereinstimmen oder der angegebene Vektor null ist.
307      */
308     public IRealMatrix leftMultiplyColumnVector(IRealColumnVector vector) {
309         if (vector == null) throw new IllegalArgumentException("vector = null.");
310         if (_rowCount != 1) throw new DimensionException();
311         if (vector instanceof RealColumnVector) {
312             // Vektor ist eine RealColumnVector-Instanz
313             double[] vectorEntries = ((RealColumnVector) vector)._entries;
```

```
314             double[] matrixEntries = this._entries; // Die Matrix hat nur eine Zeile -->
                    Eintraege entsprechen den Spalten.
315             //  Die neue Matrix hat vectorEntries.length Zeilen und _colunnCount Spalten.
316             int rowCount = vectorEntries.length;
317             int columnCount = _columnCount;
318             RealMatrix result = new RealMatrix(rowCount, columnCount);
319             double[] resultEntries = result._entries;
320             int maxIndex = resultEntries.length;
321             int currentRow = 0, currentColumn = 0, currentIndex = 0;
322             double currentFactor = vectorEntries[currentRow];
323             while (currentIndex < maxIndex) {
324                 resultEntries[currentIndex] = currentFactor * matrixEntries[currentColumn];
325                 currentColumn++;
326                 currentIndex++;
327                 if (currentColumn >= columnCount) {
328                     currentColumn = 0;
329                     currentRow++;
330                     if (currentRow < rowCount) {
331                         currentFactor = vectorEntries[currentRow];
332                     } // if (currentRow < rowCount) ...
333                 } // if (currentColumn >= columnCount) ...
334             } // while (currentIndex < maxIndex) ...
335             return result;
336         } else { // if (!(vector instanceof RealColumnVector)) ...
337             // Allgemeiner Vektor
338             double[] matrixEntries = this._entries; // Die Matrix hat nur eine Zeile -->
                    Eintraege entsprechen den Spalten.
339             // Die neue Matrix hat vector.getDimension() Zeilen und _columnCount Spalten.
340             int rowCount = vector.getDimension();
341             int columnCount = _columnCount;
342             RealMatrix result = new RealMatrix(rowCount, columnCount);
343             double[] resultEntries = result._entries;
344             int maxIndex = resultEntries.length;
345             int currentRow = 0, currentColumn = 0, currentIndex = 0;
346             double currentFactor = vector.getEntry(currentRow);
347             while (currentIndex < maxIndex) {
348                 resultEntries[currentIndex] = currentFactor * matrixEntries[currentColumn];
349                 currentColumn++;
350                 currentIndex++;
351                 if (currentColumn >= columnCount) {
352                     currentColumn = 0;
353                     currentRow++;
354                     if (currentRow < rowCount) {
355                         currentFactor = vector.getEntry(currentRow);
356                     } // if (currentRow < rowCount) ...
357                 } // if (currentColumn >= columnCount) ...
358             } // while (currentIndex < maxIndex) ...
359         } // if (!(vector instanceof RealColumnVector)) ...
360         return null;
361     } // public IRealMatrix leftMultiplyColumnVector(IRealColumnVector vector) ...
362
363     /**
364      * Multipliziert die Matrix von rechts mit dem angegebenen Spaltenvektor
365      * und gibt das Ergebnis als veraenderbaren Vektor zurueck.
366      * Die aktuelle Matrix bleibt unangetastet.
367      * @param vector Vektor, mit dem die Matrix multipliziert werden soll.
368      *   (null wird als Nullvektor interpraetiert)
369      * @return Ergebnisvektor.
370      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Wird geworfen, wenn
            die Dimensionen
371      *   nicht uebereinstimmen.
372      */
373     public IMutableRealColumnVector rightMultiplyColumnVector(IRealColumnVector vector) {
374         if (vector == null) {
375             return new RealColumnVector(_rowCount);
376         } else { // if (vector != null) ...
377             if (vector instanceof RealColumnVector) {
378                 // Vektor ist eine RealColumnVector-Instanz
379                 double[] vectorEntries = ((RealColumnVector) vector)._entries;
380                 double[] matrixEntries = this._entries;
381                 // Der Ergebnisvektor hat soviel Zeilen wie die Matrix.
382                 int rowCount = _rowCount;
383                 int columnCount = _columnCount;
384                 if (columnCount != vectorEntries.length) throw new DimensionException();
385                 RealColumnVector result = new RealColumnVector(rowCount);
386                 double[] resultEntries = result._entries;
387                 int maxIndex = matrixEntries.length;
388                 int currentRow = 0, currentColumn = 0, currentIndex = 0;
389                 while (currentIndex < maxIndex) {
```

```
390                     resultEntries[currentRow] += matrixEntries[currentIndex] * vectorEntries[
                            currentColumn];
391                     currentIndex++;
392                     currentColumn++;
393                     if (currentColumn >= columnCount) {
394                         currentColumn = 0;
395                         currentRow++;
396                     } // if (currentColumn >= columnCount) ...
397                 } // while (currentIndex < maxIndex) ...
398                 return result;
399             } else { // if (!(vector instanceof RealColumnVector)) ...
400                 // Allgemeiner Vektor
401                 double[] matrixEntries = this._entries;
402                 // Der Ergebnisvektor hat soviel Zeilen wie die Matrix.
403                 int rowCount = _rowCount;
404                 int columnCount = _columnCount;
405                 if (columnCount != vector.getDimension()) throw new DimensionException();
406                 RealColumnVector result = new RealColumnVector(rowCount);
407                 double[] resultEntries = result._entries;
408                 int maxIndex = matrixEntries.length;
409                 int currentRow = 0, currentColumn = 0, currentIndex = 0;
410                 while (currentIndex < maxIndex) {
411                     resultEntries[currentRow] += matrixEntries[currentIndex] * vector.getEntry(
                            currentColumn);
412                     currentIndex++;
413                     currentColumn++;
414                     if (currentColumn >= columnCount) {
415                         currentColumn = 0;
416                         currentRow++;
417                     } // if (currentColumn >= columnCount) ...
418                 } // while (currentIndex < maxIndex) ..
419             } // if (!(vector instanceof RealColumnVector)) ...
420             return null;
421         } // if (vector != null) ...
422     } // public IMutableRealColumnVector rightMultiplyColunVector(IRealColumnVector vector) ...
423
424     /**
425      * Multipliziert die Matrix von links mit der angegebenen Matrix
426      * und gibt das Ergebnis als Matrix zurueck. Die aktuelle Matrix bleibt
427      * unangetastet.
428      * @param matrix Matrix, mit der multipliziert werden soll.
429      * @return Ergebnismatrix.
430      * @exception IllegalArgumentException Wird geworfen, wenn die uebergebene
431      *   Matrix null ist.
432      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Wird geworfen, wenn
433      *   die Dimensionen nicht uebereinstimmen.
434      */
435     public IRealMatrix leftMultiplyMatrix(IRealMatrix matrix) {
436         if (matrix == null) throw new IllegalArgumentException("matrix = null.");
437         if (matrix instanceof RealMatrix) {
438             // RealMatrix-Instanz
439             RealMatrix realMatrix = (RealMatrix) matrix;
440             int intermediateDimension = _rowCount;
441             if (realMatrix._columnCount != intermediateDimension) throw new DimensionException
                    ();
442             // Die Ergebnis-Matrix hat die Zeilenanzahl der linken Matrix und die Spaltenanzahl
                     der rechten Matrix.
443             int rowCount = realMatrix._rowCount;
444             int columnCount = _columnCount;
445             RealMatrix result = new RealMatrix(rowCount, columnCount);
446             double[] leftEntries = realMatrix._entries;
447             double[] rightEntries = _entries;
448             double[] resultEntries = result._entries;
449             int currentRow, currentColumn, currentIntermediate, resultBaseIndex,
                    currentRightIndex;
450             double currentFactor;
451             for (currentRow = 0; currentRow < rowCount; currentRow++) {
452                 resultBaseIndex = currentRow * columnCount;
453                 currentRightIndex = 0;
454                 for (currentIntermediate = 0; currentIntermediate < intermediateDimension;
                        currentIntermediate++) {
455                     currentFactor = leftEntries[currentRow * intermediateDimension +
                            currentIntermediate];
456                     for (currentColumn = 0; currentColumn < columnCount; currentColumn++) {
457                         resultEntries[resultBaseIndex + currentColumn] += currentFactor *
                                rightEntries[currentRightIndex++];
458                     } // for (currentColumn = 0; currentColumn < columnCount; currentColumn++)
                            ...
459                 } // for (currentIntermediate = 0; currentIntermediate < intermediateDimension;
                        currentIntermediate++) ...
```

```
460             } // for (currentRow = 0; currentRow < rowCount; currentRow++) ...
461             return result;
462         } else if (matrix instanceof RealDiagonalMatrix) {
463             // RealDiagonalMatrix-Instanz
464             double[] diagonalEntries = ((RealDiagonalMatrix) matrix)._entries;
465             int rowCount = diagonalEntries.length;
466             int columnCount = _columnCount;
467             if (rowCount != _rowCount) throw new DimensionException(); // Diagonalmatrix ist
                    quadratisch --> rowCount = columnCount
468             RealMatrix result = new RealMatrix(rowCount, columnCount);
469             double[] resultEntries = result._entries;
470             double[] rightEntries = _entries;
471             int currentIndex = 0, currentColumn = 0, currentRow = 0, maxIndex = resultEntries.
                    length;
472             double currentFactor = diagonalEntries[currentRow];
473             while (currentIndex < maxIndex) {
474                 resultEntries[currentIndex] = currentFactor * rightEntries[currentIndex];
475                 currentIndex++;
476                 currentColumn++;
477                 if (currentColumn > columnCount) {
478                     currentRow++;
479                     currentColumn = 0;
480                     if (currentRow < rowCount) {
481                         currentFactor = diagonalEntries[currentRow];
482                     } // if (currentRow < rowCount) ...
483                 } // if (currentColumn > columnCount) ...
484             } // while (currentIndex < maxIndex) ...
485             return result;
486         } else if (matrix instanceof RealPermutationMatrix) {
487             // Sei A = (a_ij) eine reellwertige Matrix aus R^mxn
488             // P = (p_ij) = (delta_i_sigma(j)) eine Permutationsmatrix zur Permutation sigma
                    aus Sm
489             // und C = (c_ij) = P * A, dann gilt:
490             // c_ij = sum_{k=1..n} p_ik * a_kj = sum_{k=1..n} delta_i_sigma(k) * a_kj =
491             // = sum_{k=sigma^(-1)(i)} a_kj = a_sigma^(-1)(i)_j
492             // d.h. die i-te Zeile der Ergebnismatrix ist die sigma^(-1)(i)-te Zeile der
                    urspruenglichen Matrix.
493             double[] rightEntries = _entries;
494             int[] permutationInverseLeft = ((RealPermutationMatrix) matrix)._permutationInverse
                    ;
495             int rowCount = _rowCount;
496             int columnCount = _columnCount;
497             if (permutationInverseLeft.length != rowCount) throw new DimensionException();
498             RealMatrix result = new RealMatrix(rowCount, columnCount);
499             double[] resultEntries = result._entries;
500             int currentIndex = 0, currentColumn = 0, currentRow = 0, maxIndex = resultEntries.
                    length;
501             int baseIndex = permutationInverseLeft[currentRow] * columnCount;
502             while (currentIndex < maxIndex) {
503                 resultEntries[currentIndex] = rightEntries[baseIndex + currentColumn];
504                 currentIndex++;
505                 currentColumn++;
506                 if (currentColumn >= columnCount) {
507                     currentColumn = 0;
508                     currentRow++;
509                     if (currentRow < rowCount) {
510                         baseIndex = permutationInverseLeft[currentRow] * columnCount;
511                     } // if (currentRow < rowCount) ...
512                 } // if (currentColumn >= columnCount) ...
513             } // while (currentIndex < maxIndex) ...
514             return result;
515         } else { // Allgemeiner Fall ...
516             // Allgemeine Matrix
517             int intermediateDimension = _rowCount;
518             if (matrix.getColumnCount() != intermediateDimension) throw new DimensionException
                    ();
519             // Die Ergebnis-Matrix hat die Zeilenanzahl der linken Matrix und die Spaltenanzahl
                     der rechten Matrix.
520             int rowCount = matrix.getRowCount();
521             int columnCount = _columnCount;
522             RealMatrix result = new RealMatrix(rowCount, columnCount);
523             double[] rightEntries = _entries;
524             double[] resultEntries = result._entries;
525             int currentRow, currentColumn, currentIntermediate, resultBaseIndex,
                    currentRightIndex;
526             double currentFactor;
527             for (currentRow = 0; currentRow < rowCount; currentRow++) {
528                 resultBaseIndex = currentRow * columnCount;
529                 currentRightIndex = 0;
```

```
530                 for (currentIntermediate = 0; currentIntermediate < intermediateDimension;
                         currentIntermediate++) {
531                     currentFactor = matrix.getEntry(currentRow, currentIntermediate);
532                     for (currentColumn = 0; currentColumn < columnCount; currentColumn++) {
533                         resultEntries[resultBaseIndex + currentColumn] += currentFactor *
                                rightEntries[currentRightIndex++];
534                     } // for (currentColumn = 0; currentColumn < columnCount; currentColumn++)
                            ...
535                 } // for (currentIntermediate = 0; currentIntermediate < intermediateDimension;
                        currentIntermediate++) ...
536             } // for (currentRow = 0; currentRow < rowCount; currentRow++) ...
537             return result;
538         } // Allgemeiner Fall ...
539     } // public IRealMatrix leftMultiplyMatrix(IRealMatrix matrix) ...
540
541     /**
542      * Multipliziert die Matrix von rechts mit der angegebenen Matrix
543      * und gibt das Ergebnis als Matrix zurueck. Die aktuelle Matrix bleibt
544      * unangetastet.
545      * @param matrix Matrix, mit der multipliziert werden soll.
546      * @return Ergebnismatrix.
547      * @exception IllegalArgumentException Wird geworfen, wenn die uebergebene Matrix null ist.
548      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Wird geworfen, wenn
549      *   die Dimensionen nicht uebereinstimmen.
550      */
551     public IRealMatrix rightMultiplyMatrix(IRealMatrix matrix) {
552         if (matrix == null) throw new IllegalArgumentException("matrix = null.");
553         if (matrix instanceof RealMatrix) {
554             // RealMatrix-Instanz
555             RealMatrix realMatrix = (RealMatrix) matrix;
556             int intermediateDimension = _columnCount;
557             if (realMatrix._rowCount != intermediateDimension) throw new DimensionException();
558             // Die Ergebnis-Matrix hat die Zeilenanzahl der linken Matrix und die Spaltenanzahl
                    der rechten Matrix.
559             int rowCount = _rowCount;
560             int columnCount = realMatrix._columnCount;
561             RealMatrix result = new RealMatrix(rowCount, columnCount);
562             double[] leftEntries = _entries;
563             double[] rightEntries = realMatrix._entries;
564             double[] resultEntries = result._entries;
565             int currentColumn, currentRow, currentIntermediate, currentLeftIndex;
566             double currentResult;
567             for (currentColumn = 0; currentColumn < columnCount; currentColumn++) {
568                 currentLeftIndex = 0; // Indizes der linken Matrix koennen pro Ergebnisspalte
                        sequenziell durchlaufen werden.
569                 for (currentRow = 0; currentRow < rowCount; currentRow++) {
570                     currentResult = 0.0D;
571                     for (currentIntermediate = 0; currentIntermediate < intermediateDimension;
                             currentIntermediate++) {
572                         currentResult += leftEntries[currentLeftIndex++] * rightEntries[
                                currentIntermediate * columnCount + currentColumn];
573                     } // for (currentIntermediate = 0; currentIntermediate <
                            intermediateDimension; currentIntermediate++) ...
574                     resultEntries[currentRow * columnCount + currentColumn] = currentResult;
575                 } // for (currentRow = 0; currentRow < rowCount; currentRow++) ...
576             } // for (currentColumn = 0; currentColumn < columnCount; currentColumn++) ...
577             return result;
578         } else if (matrix instanceof RealDiagonalMatrix) {
579             // RealDiagonalMatrix-Instanz
580             double[] diagonalEntries = ((RealDiagonalMatrix) matrix)._entries;
581             int rowCount = _rowCount;
582             int columnCount = _columnCount;
583             if (diagonalEntries.length != columnCount) throw new DimensionException();
584             RealMatrix result = new RealMatrix(rowCount, columnCount);
585             double[] leftEntries = _entries;
586             double[] resultEntries = result._entries;
587             int currentIndex = 0, currentColumn = 0, maxIndex = resultEntries.length;
588             while (currentIndex < maxIndex) {
589                 resultEntries[currentIndex] = leftEntries[currentIndex] * diagonalEntries[
                        currentColumn];
590                 currentColumn++;
591                 currentIndex++;
592                 if (currentColumn >= columnCount) {
593                     currentColumn = 0;
594                 } // if (currentColumn >= columnCount) ...
595             } // while (currentIndex < maxIndex) ...
596             return result;
597         } else if (matrix instanceof RealPermutationMatrix) {
598             // Sei A = (a_ij) eine reellwertige Matrix aus R^mxn,
```

```
599             // P = (p_ij) = (delta_i_sigma(j)) eine Permutationsmatrix zur Permutation sigma
                    aus Sn
600             // und C = (c_ij) = A * P, dann gilt:
601             // c_ij = sum_{k=1..n} a_ik * p_kj = sum_{k=1..n} a_ik * delta_k_sigma(j) =
602             // = sum{k=sigma(j)} a_ik = a_i_sigma(j)
603             // d.h., die j-te-Spalte der Ergebnis-Matrix ist die sigma(j)-te Spalte der
                    urspruenglichen Matrix.
604             int rowCount = _rowCount;
605             int columnCount = _columnCount;
606             int[] permutation = ((RealPermutationMatrix) matrix)._permutation;
607             if (columnCount != permutation.length) throw new DimensionException();
608             double[] leftEntries = _entries;
609             RealMatrix result = new RealMatrix(rowCount, columnCount);
610             double[] resultEntries = result._entries;
611             int rowBaseIndex = 0, currentColumn = 0, currentIndex = 0, maxIndex = resultEntries
                    .length;
612             while (currentIndex < maxIndex) {
613                 resultEntries[currentIndex] = leftEntries[rowBaseIndex + permutation[
                        currentColumn]];
614                 currentColumn++;
615                 currentIndex++;
616                 if (currentColumn >= columnCount) {
617                     currentColumn = 0;
618                     rowBaseIndex += columnCount;
619                 } // if (currentColumn >= columnCount) ...
620             } // while (currentIndex < maxIndex) ...
621             return result;
622         } else { // Allgemeiner Fall ...
623             // Allgemeine Matrix
624             int intermediateDimension = _columnCount;
625             if (matrix.getRowCount() != intermediateDimension) throw new DimensionException();
626             // Die Ergebnis-Matrix hat die Zeilenahnzahl der linken Matrix und die
                    Spaltenanzahl der rechten Matrix.
627             int rowCount = _rowCount;
628             int columnCount = matrix.getColumnCount();
629             RealMatrix result = new RealMatrix(rowCount, columnCount);
630             double[] leftEntries = _entries;
631             double[] resultEntries = result._entries;
632             int currentColumn, currentRow, currentIntermediate, currentLeftIndex;
633             double currentResult;
634             for (currentColumn = 0; currentColumn < columnCount; currentColumn++) {
635                 currentLeftIndex = 0; // Indizes der linken Matrix koennen pro Ergebnisspalte
                        sequenziell durchlaufen werden.
636                 for (currentRow = 0; currentRow < rowCount; currentRow++) {
637                     currentResult = 0.0D;
638                     for (currentIntermediate = 0; currentIntermediate < intermediateDimension;
                            currentIntermediate++) {
639                         currentResult += leftEntries[currentLeftIndex++] * matrix.getEntry(
                                currentIntermediate, currentColumn);
640                     } // for (currentIntermediate = 0; currentIntermediate <
                            intermediateDimension; currentIntermediate++) ...
641                     resultEntries[currentRow * columnCount + currentRow] = currentResult;
642                 } // for (currentRow = 0; currentRow < rowCount; currentRow++) ...
643             } // for (currentColumn = 0; currentColumn < columnCount; currentColumn++) ...
644         } // Allgemeiner Fall ...
645         return null;
646     } // public IRealMatrix rightMultiplyMatrix(IRealMatrix matrix) ...
647
648     /**
649      * Setzt den Eintrag an der angegebenen Position
650      * auf den angegebenen Wert.
651      * @param row Zeile des Eintrags (Nullbasiert).
652      * @param column Spalte des Eintrags (Nullbasiert).
653      * @param value Wert, auf den der Eintrag gesetzt werden soll.
654      * @exception IndexOutOfBoundsException Wird geworfen, wenn die
655      *   angegebene Position nicht existiert.
656      */
657     public void setEntry(int row, int column, double value) {
658         if (row < 0 || row >= _rowCount) throw new IndexOutOfBoundsException("row");
659         int columnCount = _columnCount;
660         if (column < 0 || column >= columnCount) throw new IndexOutOfBoundsException("column");
661         _entries[row * columnCount + column] = value;
662     } // public void setEntry(int row, int column, double value) ...
663
664     /**
665      * Addiert zum Eintrag an der angegebenen Position
666      * den angegebenen Wert.
667      * @param row Zeile des Eintrags (Nullbasiert).
668      * @param column Spalte des Eintrags (Nullbasiert).
669      * @param value Wert, der addiert werden soll.
```

```
670      * @exception IndexOutOfBoundsException Wird geworfen, wenn die
671      *   angegebene Position nicht existiert.
672      */
673     public void addEntry(int row, int column, double value) {
674         if (row < 0 || row >= _rowCount) throw new IndexOutOfBoundsException("row");
675         int columnCount = _columnCount;
676         if (column < 0 || column >= columnCount) throw new IndexOutOfBoundsException("column");
677         _entries[row * columnCount + column] += value;
678     } // public void addEntry(int row, int column, double value) ...
679 
680     /**
681      * Multipliziert den Eintrag an der angegebenen Position
682      * mit dem angegebenen Faktor.
683      * @param row Zeile des Eintrags (Nullbasiert).
684      * @param column Spalte des Eintrags (Nullbasiert).
685      * @param factor Faktor, mit dem der Eintrag multipliziert werden soll.
686      * @exception IndexOutOfBoundsException Wird geworfen, wenn die
687      *   angegebene Position nicht existiert.
688      */
689     public void multiplyEntry(int row, int column, double factor) {
690         if (row < 0 || row >= _rowCount) throw new IndexOutOfBoundsException("row");
691         int columnCount = _columnCount;
692         if (column < 0 || column >= _columnCount) throw new IndexOutOfBoundsException("column")
                 ;
693         _entries[row * columnCount + column] *= factor;
694     } // public void multiplyEntry(int row, int column, double factor) ...
695 } // public class RealMatrix implements IMutableRealMatrix ...
```

Listing 24: Klasse *at.techmath.boltzmann.linearalgebra.RealPermutationMatrix*

```
1  package at.techmath.boltzmann.linearalgebra;
2
3  /**
4   * Repraesentiert eine reellwertige Permutationsmatrix.
5   * Zu einer Permutation sigma aus Sn gilt mit P = (p_ij) p_ij = delta_i_sigma(j).
6   * @author Stefan Schnabl (e0226245)
7   */
8  public class RealPermutationMatrix implements IRealPermutationMatrix {
9      /**
10      * Erzeugt eine neu Permutationsmatrix, die auf die
11      * Identitaetsmatrix initialisiert ist.
12      * @param dimension Dimension der Matrix.
13      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Die Dimension ist
            kleiner als 1.
14      */
15     public RealPermutationMatrix(int dimension) {
16         if (dimension < 1) throw new DimensionException("Die Dimension ist kleiner als 1.");
17         _permutation = new int[dimension];
18         _permutationInverse = new int[dimension];
19         // Permutation und ihre Inverse auf die Identitaet initialisieren.
20         for (int currentIndex = 0; currentIndex < dimension; currentIndex++) {
21             _permutation[currentIndex] = currentIndex;
22             _permutationInverse[currentIndex] = currentIndex;
23         } // for (int currentIndex = 0; currentIndex < dimension; currentIndex++) ...
24     } // public RealPermutationMatrix(int dimension) ...
25
26     /**
27      * Gibt eine String-Repraesentation der Matrix zurueck.
28      * @return String-Repraesentation der Matrix.
29      */
30     @Override
31     public String toString() {
32         int dimension = _permutationInverse.length;
33         StringBuilder sb = new StringBuilder();
34         sb.append('[');
35         for (int currentRow = 0; currentRow < dimension; currentRow++) {
36             if (currentRow > 0) {
37                 sb.append(',');
38             } // if (currentRow > 0) ...
39             sb.append('(');
40             int currentColumnOne = _permutationInverse[currentRow]; // Index der Spalte, in der
                    in der aktuellen Zeile der Einser steht
41             for (int currentColumn = 0; currentColumn < dimension; currentColumn++) {
42                 if (currentColumn > 0) {
43                     sb.append(',');
44                 } // if (currentColumn > 0) ...
45                 if (currentColumn == currentColumnOne) {
46                     sb.append(1.0D);
47                 } else { // if (currentColumn != currentColumnOne) ...
48                     sb.append(0.0D);
49                 } // if (currentColumn != currentColumnOne) ...
50             } // for (int currentColumn = 0; currentColumn < dimension; currentColumn++) ...
51             sb.append(')');
52         } // for (int currentRow = 0; currentRow < dimension; currentRow++) ...
53         sb.append(']');
54         return sb.toString();
55     } // public String toString() ...
56
57     /**
58      * Einzeilige Darstellung der Permutation.
59      * p(i) = _permutation[i] fuer i = 0 .. dimension - 1.
60      * p(i) gibt zur Spalte i die Zeile an, in der in der Spalte der Einser steht.
61      */
62     protected int[] _permutation;
63
64     /**
65      * Einzeilige Darstellung der inversen Permutation.
66      * p^(-1) (i) = _permutationInverse[i] fuer i = 0 .. dimension - 1.
67      * p^(-1) (i) gibt zur Zeile i die Spalte an, in der in der Zeile der Einser steht.
68      */
69     protected int[] _permutationInverse;
70
71     /**
72      * Tauscht die beiden angegebenen Spalten aus.
73      * @param firstColumn Index der ersten Spalte.
74      * @param secondColumn Index der zweiten Spalte.
75      * @exception IndexOutOfBoundsException Wird geworfen, wenn mindestens
76      *   einer der beiden angegebenen Indizes ausserhalb des gueltigen Bereichs ist.
77      */
```

```
78      public void swapColumns(int firstColumn, int secondColumn) {
79          int dimension = _permutation.length;
80          if (firstColumn < 0 || firstColumn >= dimension) throw new IndexOutOfBoundsException("
                firstColumn");
81          if (secondColumn < 0 || secondColumn >= dimension) throw new IndexOutOfBoundsException(
                "secondColumn");
82          if (firstColumn != secondColumn) {
83              // Relevante Zeilen bestimmen
84              int firstRow = _permutation[firstColumn];
85              int secondRow = _permutation[secondColumn];
86              int t;
87              // Relevanten Zeilen in der inversen Permutation austauschen
88              t = _permutationInverse[firstRow];
89              _permutationInverse[firstRow] = _permutationInverse[secondRow];
90              _permutationInverse[secondRow] = t;
91              // Relevante Spalten in Permutation austauschen
92              t = _permutation[firstColumn];
93              _permutation[firstColumn] = _permutation[secondColumn];
94              _permutation[secondColumn] = t;
95          } // if (firstColumn != secondColumn) ...
96      } // public void swapColumns(int firstColumn, int secondColumn) ...
97
98      /**
99       * Tauscht die beiden angegebenen Zeilen aus.
100      * @param firstRow Index der ersten Zeile.
101      * @param secondRow Index der zweiten Zeile.
102      * @exception IndexOutOfBoundsException Wird geworfen, wenn mindestens
103      *   einer der beiden angegebenen Indizes ausserhalb des gueltigen Bereichs ist.
104      */
105     public void swapRows(int firstRow, int secondRow) {
106         int dimension = _permutation.length;
107         if (firstRow < 0 || firstRow >= dimension) throw new IndexOutOfBoundsException("
                firstRow");
108         if (secondRow < 0 || secondRow >= dimension) throw new IndexOutOfBoundsException("
                secondRow");
109         if (firstRow != secondRow) {
110             // Relevante Spalten bestimmen
111             int firstColumn = _permutationInverse[firstRow];
112             int secondColumn = _permutationInverse[secondRow];
113             int t;
114             // Relevante Spalten in der Permutation austauschen
115             t = _permutation[firstColumn];
116             _permutation[firstColumn] = _permutation[secondColumn];
117             _permutation[secondColumn] = t;
118             // Relevante Zeilen in der inversen Permutation austauschen
119             t = _permutationInverse[firstRow];
120             _permutationInverse[firstRow] = _permutationInverse[secondRow];
121             _permutationInverse[secondRow] = t;
122         } // if (firstRow != secondRow) ...
123     } // public void swapRows(int firstRow, int secondRow) ...
124
125     /**
126      * Gibt die Dimension der Matrix zurueck.
127      * @return Dimension der Matrix.
128      */
129     public int getDimension() {
130         return _permutation.length;
131     } // public int getDimension() ...
132
133     /**
134      * Gibt die Anzahl der Zeilen der Matrix zurueck.
135      * @return Anzahl der Zeilen der Matrix.
136      */
137     public int getRowCount() {
138         return _permutation.length;
139     } // public int getRowCount() ...
140
141     /**
142      * Gibt die Anzahl der Spalten der Matrix zurueck.
143      * @return Anzahl der Spalten der Matrix.
144      */
145     public int getColumnCount() {
146         return _permutation.length;
147     } // public int getColumnCount() ...
148
149     /**
150      * Gibt zurueck, ob es sich um eine quadratische Matrix handelt.
151      * @return true, wenn es sich um eine quadratische Matrix handelt, ansonsten false.
152      */
153     public boolean isSquare() {
```

```
154         return true;
155     } // public boolean isSquare() ...
156 
157     /**
158      * Gibt den Eintrag an der angegebenen Position zurueck.
159      * @param row Zeile des Eintrags (Nullbasiert).
160      * @param column Spalte des Eintrags (Nullbasiert).
161      * @return Eintrag an der angegebenen Position.
162      * @exception IndexOutOfBoundsException Wird geworfen, wenn die angegebene
163      *   Position nicht existiert.
164      */
165     public double getEntry(int row, int column) {
166         int dimension = _permutation.length;
167         if (row < 0 || row >= dimension) throw new IndexOutOfBoundsException("row");
168         if (column < 0 || column >= dimension) throw new IndexOutOfBoundsException("column");
169         return row == _permutation[column] ? 1.0D : 0.0D;
170     } // public double getEntry(int row, int column) ...
171 
172     /**
173      * Gibt eine veraenderbare Kopie der aktuellen Matrix zurueck.
174      * @return Veraenderbare Kopie der aktuellen Matrix.
175      */
176     public IMutableRealMatrix getMutableCopy() {
177         int dimension = _permutation.length;
178         RealMatrix result = new RealMatrix(dimension, dimension);
179         double[] resultEntries = result._entries;
180         for (int currentColumn = 0; currentColumn < dimension; currentColumn++) {
181             int currentRow = _permutation[currentColumn];
182             resultEntries[currentRow * dimension + currentColumn] = 1.0D;
183         } // for (int currentColumn = 0; currentColumn < dimension; currentColumn++) ...
184         return result;
185     } // public IMutableRealMatrix getMutableCopy() ...
186 
187     /**
188      * Multipliziert die Matrix von links mit dem angegebenen Zeilenvektor
189      * und gibt das Ergebnis als veraenderbaren Vektor zurueck.
190      * Die aktuelle Matrix bleibt unangetastet.
191      * @param vector Vektor, mit dem die Matrix multipliziert werden soll.
192      *   (null wird als Nullvektor interpraetiert)
193      * @return Ergebnisvektor.
194      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Wird geworfen, wenn
         die Dimensionen
195      *   nicht uebereinstimmen.
196      */
197     public IMutableRealRowVector leftMultiplyRowVector(IRealRowVector vector) {
198         if (vector == null) {
199             return new RealRowVector(_permutation.length);
200         } else { // if (vector != null) ...
201             if (vector instanceof RealRowVector) {
202                 int dimension = _permutation.length;
203                 int[] permutation = _permutation;
204                 double[] vectorEntries = ((RealRowVector) vector)._entries;
205                 if (vectorEntries.length != dimension) throw new DimensionException();
206                 RealRowVector result = new RealRowVector(dimension);
207                 double[] resultEntries = result._entries;
208                 for (int currentColumn = 0; currentColumn < dimension; currentColumn++) {
209                     resultEntries[currentColumn] = vectorEntries[permutation[currentColumn]];
210                 } // for (int currentColumn = 0; currentColumn < dimension; currentColumn++)
                      ...
211                 return result;
212             } else { // if (!(vector instanceof RealRowVector)) ...
213                 int dimension = _permutation.length;
214                 int[] permutation = _permutation;
215                 if (vector.getDimension() != dimension) throw new DimensionException();
216                 RealRowVector result = new RealRowVector(dimension);
217                 double[] resultEntries = result._entries;
218                 for (int currentColumn = 0; currentColumn < dimension; currentColumn++) {
219                     resultEntries[currentColumn] = vector.getEntry(permutation[currentColumn]);
220                 } // for (int currentColumn = 0; currentColumn < dimension; currentColumn++)
                      ...
221                 return result;
222             } // if (!(vector instanceof RealRowVector)) ...
223         } // if (vector != null) ...
224     } // public IMutableRealRowVector leftMultiplyRowVector(IRealRowVector vector) ...
225 
226     /**
227      * Multipliziert die Matrix von rechts mit dem angegebenen Zeilenvektor
228      * und gibt das Ergebnis als Matrix zurueck. Die aktuelle Matrix bleibt
229      * unangetastet.
230      * @param vector Vektor, mit dem die Matrix multipliziert werden soll.
```

```
231      * @return Ergebnismatrix.
232      * @exception IllegalArgumentException Wird geworfen, wenn der angegebene Vektor null ist.
233      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Wird geworfen, wenn
             die Dimensionen
234      *   nicht uebereinstimmen.
235      */
236     public IRealMatrix rightMultiplyRowVector(IRealRowVector vector) {
237         if (vector == null) throw new IllegalArgumentException("vector = null.");
238         if (_permutation.length != 1) throw new DimensionException();
239         // Damit die Dimensionen passen, muss die Matrix die Dimension 1 besitzen.
240         // Eine Permutationsmatrix mit Dimension 1 hat genau einen Eintrag, der 1 ist.
241         // Daher ist das Ergebnis der Multiplikation der Vektor selber.
242         if (vector instanceof RealRowVector) {
243             double[] vectorEntries = ((RealRowVector) vector)._entries;
244             int columnCount = vectorEntries.length;
245             RealMatrix result = new RealMatrix(1, columnCount);
246             double[] resultEntries = result._entries;
247             for (int currentColumn = 0; currentColumn < columnCount; currentColumn++) {
248                 resultEntries[currentColumn] = vectorEntries[currentColumn];
249             } // for (int currentColumn = 0; currentColumn < columnCount; currentColumn++) ...
250             return result;
251         } else { // if (!(vector instanceof RealRowVector)) ...
252             int columnCount = vector.getDimension();
253             RealMatrix result = new RealMatrix(1, columnCount);
254             double[] resultEntries = result._entries;
255             for (int currentColumn = 0; currentColumn < columnCount; currentColumn++) {
256                 resultEntries[currentColumn] = vector.getEntry(currentColumn);
257             } // for (int currentColumn = 0; currentColumn < columnCount; currentColumn++) ...
258             return result;
259         } // if (!(vector instanceof RealRowVector)) ...
260     } // public IRealMatrix rightMultiplyRowVector(IRealRowVector vector) ...
261
262     /**
263      * Multipliziert die Matrix von links mit dem angegebenen Spaltenvektor
264      * und gibt das Ergebnis als Matrix zurueck. Die aktuelle Matrix bleibt
265      * unangetastet.
266      * @param vector Vektor, mit dem die Matrix multipliziert werden soll.
267      * @return Ergebnismatrix.
268      * @exception IllegalArgumentException Wird geworfen, wenn der angegebene Vektor null ist.
269      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Wird geworfen, wenn
             die Dimensionen
270      *   nicht uebereinstimmen.
271      */
272     public IRealMatrix leftMultiplyColumnVector(IRealColumnVector vector) {
273         if (vector == null) throw new IllegalArgumentException("vector = null.");
274         if (_permutation.length != 1) throw new DimensionException();
275         // Damit die Dimensionen passen, muss die Matrix die Dimension 1 besitzen.
276         // Eine Permutationsmatrix mit Dimension 1 hat genau einen Eintrag, der 1 ist.
277         // Daher ist das Ergebnis der Multiplikation der Vektor selber.
278         if (vector instanceof RealColumnVector) {
279             double[] vectorEntries = ((RealColumnVector) vector)._entries;
280             int rowCount = vectorEntries.length;
281             RealMatrix result = new RealMatrix(rowCount, 1);
282             double[] resultEntries = result._entries;
283             for (int currentRow = 0; currentRow < rowCount; currentRow++) {
284                 resultEntries[currentRow] = vectorEntries[currentRow];
285             } // for (int currentRow = 0; currentRow < rowCount; currentRow++) ...
286             return result;
287         } else { // if (!(vector instanceof RealColumnVector)) ...
288             int rowCount = vector.getDimension();
289             RealMatrix result = new RealMatrix(rowCount, 1);
290             double[] resultEntries = result._entries;
291             for (int currentRow = 0; currentRow < rowCount; currentRow++) {
292                 resultEntries[currentRow] = vector.getEntry(currentRow);
293             } // for (int currentRow = 0; currentRow < rowCount; currentRow++) ...
294             return result;
295         } // if (!(vector instanceof RealColumnVector)) ...
296     } // public IRealMatrix leftMultiplyColumnVector(IRealColumnVector vector) ...
297
298     /**
299      * Multipliziert die Matrix von rechts mit dem angegebenen Spaltenvektor
300      * und gibt das Ergebnis als veraenderbaren Vektor zurueck.
301      * Die aktuelle Matrix bleibt unangetastet.
302      * @param vector Vektor, mit dem die Matrix multipliziert werden soll.
303      *   (null wird als Nullvektor interpraetiert)
304      * @return Ergebnisvektor.
305      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Wird geworfen, wenn
             die Dimensionen
306      *   nicht uebereinstimmen.
307      */
```

```
308     public IMutableRealColumnVector rightMultiplyColumnVector(IRealColumnVector vector) {
309         if (vector == null) {
310             // Wenn null uebergeben wird, wird als Argument der Nullvektor mit der
311             // passenden Dimension verwendet.
312             return new RealColumnVector(_permutation.length);
313         } else { // if (vector != null) ...
314             if (vector instanceof RealColumnVector) {
315                 int[] permutationInverse = _permutationInverse;
316                 int dimension = permutationInverse.length;
317                 double[] vectorEntries = ((RealColumnVector) vector)._entries;
318                 if (vectorEntries.length != dimension) throw new DimensionException();
319                 RealColumnVector result = new RealColumnVector(dimension);
320                 double[] resultEntries = result._entries;
321                 for (int currentRow = 0; currentRow < dimension; currentRow++) {
322                     resultEntries[currentRow] = vectorEntries[permutationInverse[currentRow]];
323                 } // for (int currentRow = 0; currentRow < dimension; currentRow++) ...
324                 return result;
325             } else { // if (!(vector instanceof RealColumnVector)) ...
326                 int[] permutationInverse = _permutationInverse;
327                 int dimension = permutationInverse.length;
328                 if (vector.getDimension() != dimension) throw new DimensionException();
329                 RealColumnVector result = new RealColumnVector(dimension);
330                 double[] resultEntries = result._entries;
331                 for (int currentRow = 0; currentRow < dimension; currentRow++) {
332                     resultEntries[currentRow] = vector.getEntry(permutationInverse[currentRow])
                            ;
333                 } // for (int currentRow = 0; currentRow < dimension; currentRow++) ...
334                 return result;
335             } // if (!(vector instanceof RealColumnVector)) ...
336         } // if (vector != null) ...
337     } // public IMutableRealColumnVector rightMultiplyColumnVector(IRealColumnVector vector)
         ...
338
339     /**
340      * Multipliziert die Matrix von links mit der angegebenen Matrix
341      * und gibt das Ergebnis als Matrix zurueck. Die aktuelle Matrix bleibt
342      * unangetastet.
343      * @param matrix Matrix, mit der multipliziert werden soll.
344      * @return Ergebnismatrix.
345      * @exception IllegalArgumentException Wird geworfen, wenn die uebergebene Matrix null ist.
346      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Wird geworfen, wenn
           die Dimensionen
347      *   nicht uebereinstimmen.
348      */
349     public IRealMatrix leftMultiplyMatrix(IRealMatrix matrix) {
350         if (matrix == null) throw new IllegalArgumentException("matrix = null.");
351         if (matrix instanceof RealMatrix) {
352             // Sei A = (a_ij) eine reellwertige Matrix aus R^mxn,
353             // P = (p_ij) = (delta_i_sigma(j)) eine Permutationsmatrix zur Permutation sigma
                    aus Sn
354             // und C = (c_ij) = A * P, dann gilt:
355             // c_ij = sum_{k=1..n} a_ik * p_kj = sum_{k=1..n} a_ik * delta_k_sigma(j) =
356             // = sum{k=sigma(j)} a_ik = a_i_sigma(j)
357             // d.h., die j-te-Spalte der Ergebnis-Matrix ist die sigma(j)-te Spalte der
                    urspruenglichen Matrix.
358             RealMatrix leftMatrix = (RealMatrix) matrix;
359             int rowCount = leftMatrix._rowCount;
360             int columnCount = leftMatrix._columnCount;
361             int[] permutation = _permutation;
362             if (columnCount != permutation.length) throw new DimensionException();
363             double[] leftEntries = leftMatrix._entries;
364             RealMatrix result = new RealMatrix(rowCount, columnCount);
365             double[] resultEntries = result._entries;
366             int rowBaseIndex = 0, currentColumn = 0, currentIndex = 0, maxIndex = resultEntries
                    .length;
367             while (currentIndex < maxIndex) {
368                 resultEntries[currentIndex] = leftEntries[rowBaseIndex + permutation[
                        currentColumn]];
369                 currentColumn++;
370                 currentIndex++;
371                 if (currentColumn >= columnCount) {
372                     currentColumn = 0;
373                     rowBaseIndex += columnCount;
374                 } // if (currentColumn >= columnCount) ...
375             } // while (currentIndex < maxIndex) ...
376             return result;
377         } else if (matrix instanceof RealDiagonalMatrix) {
378             // Sei D = (d_ij) = (dj delta_i_j) eine Diagonalmatrix,
379             // P = (p_ij) = (delta_i_sigma(j)) eine Permutationsmatrix zur Permutation sigma
                    aus Sn
```

```
380             // und C = (c_ij) = D * P, dann gilt:
381             // c_ij = sum_{k=1..n} d_ik * p_kj = sum_{k=1..n} dk * delta_i_k * delta_k_sigma(j)
                    =
382             // = di * delta_i_sigma(j) =
383             // = { di falls i = sigma(j), 0 sonst } =
384             // = { di falls j = sigma^(-1)(i), 0 sonst }
385             // also steht in der i-ten Zeile der Ergebnis-Matrix der i-te Diagonaleintrag in
                    der sigma^(-1)(i)-ten Spalte.
386             double[] leftDiagonal = ((RealDiagonalMatrix) matrix)._entries;
387             int[] permutationInverseRight = _permutationInverse;
388             int dimension = permutationInverseRight.length;
389             if (leftDiagonal.length != dimension) throw new DimensionException();
390             RealMatrix result = new RealMatrix(dimension, dimension);
391             double[] resultEntries = result._entries;
392             for (int currentRow = 0; currentRow < dimension; currentRow++) {
393                 resultEntries[currentRow * dimension + permutationInverseRight[currentRow]] =
                        leftDiagonal[currentRow];
394             } // for (int currentRow = 0; currentRow < dimension; currentRow++) ...
395             return result;
396         } else if (matrix instanceof RealPermutationMatrix) {
397             // Seien sigma und tau zwei Permutationen aus Sn
398             // A = (a_ij) = delta_i_sigma(j) und B = (b_ij) = delta_i_tau(j)
399             // C = A * B = (c_ij)
400             // --> c_ij = sum_{k=1..n} a_ik * b_kj = sum_{k=1..n} delta_i_sigma(k) *
                    delta_k_tau(j)
401             // = sum_{k=1..n} { 1 fuer i = sigma(k) und k = tau(j), 0 sonst }
402             // = sum_{k=tau(j)} { 1 fuer i = sigma(k), 0 sonst }
403             // = delta_i_simga(tau(j)) = delta_i_(sigma nach tau)(j)
404             // Also uebersetzt sich die Matrix-Multiplikation zweier Permutationsmatrizen
405             // in die Hintereinanderausfuehrung der Permutationen.
406             RealPermutationMatrix leftMatrix = (RealPermutationMatrix) matrix;
407             int[] permutationLeft = leftMatrix._permutation;
408             int[] permutationInverseLeft = leftMatrix._permutationInverse;
409             int[] permutationRight = _permutation;
410             int[] permutationInverseRight = _permutationInverse;
411             int dimension = permutationLeft.length;
412             if (permutationRight.length != dimension) throw new DimensionException();
413             RealPermutationMatrix result = new RealPermutationMatrix(dimension);
414             int[] permutationResult = result._permutation;
415             int[] permutationInverseResult = result._permutationInverse;
416             for (int currentIndex = 0; currentIndex < dimension; currentIndex++) {
417                 permutationResult[currentIndex] = permutationLeft[permutationRight[currentIndex
                        ]];
418                 permutationInverseResult[currentIndex] = permutationInverseRight[
                        permutationInverseLeft[currentIndex]];
419             } // for (int currentIndex = 0; currentIndex < dimension; currentIndex++) ...
420             return result;
421         } else { // Allgemeine Matrix
422             // Sei A = (a_ij) eine reellwertige Matrix aus R^mxn,
423             // P = (p_ij) = (delta_i_sigma(j)) eine Permutationsmatrix zur Permutation sigma
                    aus Sn
424             // und C = (c_ij) = A * P, dann gilt:
425             // c_ij = sum_{k=1..n} a_ik * p_kj = sum_{k=1..n} a_ik * delta_k_sigma(j) =
426             // = sum{k=sigma(j)} a_ik = a_i_sigma(j)
427             // d.h., die j-te-Spalte der Ergebnis-Matrix ist die sigma(j)-te Spalte der
                    urspruenglichen Matrix.
428             int rowCount = matrix.getRowCount();
429             int columnCount = matrix.getColumnCount();
430             int[] permutation = _permutation;
431             if (columnCount != permutation.length) throw new DimensionException();
432             RealMatrix result = new RealMatrix(rowCount, columnCount);
433             double[] resultEntries = result._entries;
434             int currentColumn = 0, currentRow = 0, currentIndex = 0, maxIndex = resultEntries.
                    length;
435             while (currentIndex < maxIndex) {
436                 resultEntries[currentIndex] = matrix.getEntry(currentRow, permutation[
                        currentColumn]);
437                 currentColumn++;
438                 currentIndex++;
439                 if (currentColumn >= columnCount) {
440                     currentColumn = 0;
441                     currentRow++;
442                 } // if (currentColumn >= columnCount) ...
443             } // while (currentIndex < maxIndex) ...
444             return result;
445         } // Allgemeine Matrix ...
446     } // public IRealMatrix leftMultiplyMatrix(IRealMatrix matrix) ...
447
448     /**
449      * Multipliziert die Matrix von rechts mit der angegebenen Matrix
```

```
450      * und gibt das Ergebnis als Matrix zurueck. Die aktuelle Matrix bleibt
451      * unangetastet.
452      * @param matrix Matrix, mit der multipliziert werden soll.
453      * @return Ergebnismatrix.
454      * @exception IllegalArgumentException Wird geworfen, wenn die uebergebene Matrix null ist.
455      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Wird geworfen, wenn
             die Dimensionen
456      *   nicht uebereinstimmen.
457      */
458     public IRealMatrix rightMultiplyMatrix(IRealMatrix matrix) {
459         if (matrix == null) throw new IllegalArgumentException("matrix = null.");
460         if (matrix instanceof RealMatrix) {
461             // Sei A = (a_ij) eine reellwertige Matrix aus R^mxn
462             // P = (p_ij) = (delta_i_sigma(j)) eine Permutationsmatrix zur Permutation sigma
                    aus Sm
463             // und C = (c_ij) = P * A, dann gilt:
464             // c_ij = sum_{k=1..n} p_ik * a_kj = sum_{k=1..n} delta_i_sigma(k) * a_kj =
465             // = sum_{k=sigma^(-1)(i)} a_kj = a_sigma^(-1)(i)_j
466             // d.h. die i-te Zeile der Ergebnismatrix ist die sigma^(-1)(i)-te Zeile der
                    urspruenglichen Matrix.
467             RealMatrix rightMatrix = (RealMatrix) matrix;
468             double[] rightEntries = rightMatrix._entries;
469             int[] permutationInverseLeft = _permutationInverse;
470             int rowCount = rightMatrix._rowCount;
471             int columnCount = rightMatrix._columnCount;
472             if (permutationInverseLeft.length != rowCount) throw new DimensionException();
473             RealMatrix result = new RealMatrix(rowCount, columnCount);
474             double[] resultEntries = result._entries;
475             int currentIndex = 0, currentColumn = 0, currentRow = 0, maxIndex = resultEntries.
                    length;
476             int baseIndex = permutationInverseLeft[currentRow] * columnCount;
477             while (currentIndex < maxIndex) {
478                 resultEntries[currentIndex] = rightEntries[baseIndex + currentColumn];
479                 currentIndex++;
480                 currentColumn++;
481                 if (currentColumn >= columnCount) {
482                     currentColumn = 0;
483                     currentRow++;
484                     if (currentRow < rowCount) {
485                         baseIndex = permutationInverseLeft[currentRow] * columnCount;
486                     } // if (currentRow < rowCount) ...
487                 } // if (currentColumn >= columnCount) ...
488             } // while (currentIndex < maxIndex) ...
489             return result;
490         } else if (matrix instanceof RealDiagonalMatrix) {
491             // Sei D = (d_ij) = (di delta_i_j) eine Diagonalmatrix,
492             // P = (p_ij) = (delta_i_sigma(j)) eine Permutationsmatrix zur Permutation sigma
                    aus Sn
493             // und C = (c_ij) = P * D, dann gilt:
494             // c_ij = sum_{k=1..n} p_ik * d_kj = sum_{k=1..n} dk * delta_i_sigma(k) * delta_k_j
                    =
495             // = sum_{k=j} dk * delta_i_sigma(k) = dj * delta_i_sigma(j) =
496             // = { dj falls i = sigma(j), 0 sonst }
497             // also steht in der j-ten Spalte der Ergebnis-Matrix der j-te Diagonaleintrag in
                    der sigma(j)-ten Zeile.
498             double[] rightDiagonal = ((RealDiagonalMatrix) matrix)._entries;
499             int[] permutationLeft = _permutation;
500             int dimension = permutationLeft.length;
501             if (rightDiagonal.length != dimension) throw new DimensionException();
502             RealMatrix result = new RealMatrix(dimension, dimension);
503             double[] resultEntries = result._entries;
504             for (int currentColumn = 0; currentColumn < dimension; currentColumn++) {
505                 resultEntries[permutationLeft[currentColumn] * dimension + currentColumn] =
                        rightDiagonal[currentColumn];
506             } // for (int currentColum = 0; currentColumn < dimension; currentColumn++) ...
507             return result;
508         } else if (matrix instanceof RealPermutationMatrix) {
509             // Seien sigma und tau zwei Permutationen aus Sn
510             // A = (a_ij) = delta_i_sigma(j) und B = (b_ij) = delta_i_tau(j)
511             // C = A * B = (c_ij)
512             // --> c_ij = sum_{k=1..n} a_ik * b_kj = sum_{k=1..n} delta_i_sigma(k) *
                    delta_k_tau(j)
513             // = sum_{k=1..n} { 1 fuer i = sigma(k) und k = tau(j), 0 sonst }
514             // = sum_{k=tau(j)} { 1 fuer i = sigma(k), 0 sonst }
515             // = delta_i_simga(tau(j)) = delta_i_(sigma nach tau)(j)
516             // Also uebersetzt sich die Matrix-Multiplikation zweier Permutationsmatrizen
517             // in die Hintereinanderausfuehrung der Permutationen.
518             RealPermutationMatrix rightMatrix = (RealPermutationMatrix) matrix;
519             int[] permutationLeft = _permutation;
520             int[] permutationInverseLeft = _permutationInverse;
```

```
521             int[] permutationRight = rightMatrix._permutation;
522             int[] permutationInverseRight = rightMatrix._permutationInverse;
523             int dimension = permutationLeft.length;
524             if (permutationRight.length != dimension) throw new DimensionException();
525             RealPermutationMatrix result = new RealPermutationMatrix(dimension);
526             int[] permutationResult = result._permutation;
527             int[] permutationInverseResult = result._permutationInverse;
528             for (int currentIndex = 0; currentIndex < dimension; currentIndex++) {
529                 permutationResult[currentIndex] = permutationLeft[permutationRight[currentIndex
                        ]];
530                 permutationInverseResult[currentIndex] = permutationInverseRight[
                        permutationInverseLeft[currentIndex]];
531             } // for (int currentIndex = 0; currentIndex < dimension; currentIndex++) ...
532             return result;
533         } else { // Allgemeine Matrix ...
534             // Sei A = (a_ij) eine reellwertige Matrix aus R^mxn
535             // P = (p_ij) = (delta_i_sigma(j)) eine Permutationsmatrix zur Permutation sigma
                    aus Sm
536             // und C = (c_ij) = P * A, dann gilt:
537             // c_ij = sum_{k=1..n} p_ik * a_kj = sum_{k=1..n} delta_i_sigma(k) * a_kj =
538             // = sum_{k=sigma^(-1)(i)} a_kj = a_sigma^(-1)(i)_j
539             // d.h. die i-te Zeile der Ergebnismatrix ist die sigma^(-1)(i)-te Zeile der
                    urspruenglichen Matrix.
540             int[] permutationInverseLeft = _permutationInverse;
541             int rowCount = matrix.getRowCount();
542             int columnCount = matrix.getColumnCount();
543             if (permutationInverseLeft.length != rowCount) throw new DimensionException();
544             RealMatrix result = new RealMatrix(rowCount, columnCount);
545             double[] resultEntries = result._entries;
546             int currentIndex = 0, currentColumn = 0, currentRow = 0, maxIndex = resultEntries.
                    length;
547             int rightRow = permutationInverseLeft[currentRow];
548             while (currentIndex < maxIndex) {
549                 resultEntries[currentIndex] = matrix.getEntry(rightRow, currentColumn);
550                 currentIndex++;
551                 currentColumn++;
552                 if (currentColumn >= columnCount) {
553                     currentColumn = 0;
554                     currentRow++;
555                     if (currentRow < rowCount) {
556                         rightRow = permutationInverseLeft[currentRow];
557                     } // if (currentRow < rowCount) ...
558                 } // if (currentColumn >= columnCount) ...
559             } // while (currentIndex < maxIndex) ...
560             return result;
561         } // Allgemeine Matrix ...
562     } // public IRealMatrix rightMultiplyMatrix(IRealMatrix matrix) ...
563 } // public class RealPermutationMatrix implements IRealPermutatinMatrix ...
```

Listing 25: Klasse *at.techmath.boltzmann.linearalgebra.RealRowVector*

```
1  package at.techmath.boltzmann.linearalgebra;
2
3  /**
4   * Die RealRowVector-Klasse stellt einen reellwertigen Zeilenvektor dar.
5   * @author Stefan Schnabl (e0226245)
6   */
7  public class RealRowVector implements IMutableRealRowVector {
8      /**
9       * Erstellt eine neue RealRowVector-Instanz, die
10      * mit 0 initialisiert ist, und die angegebene Dimension besitzt.
11      * @param dimension Dimension des Vektors.
12      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Wird geworfen, wenn
            dimension < 1 ist.
13      */
14     public RealRowVector(int dimension) {
15         if (dimension < 1) throw new DimensionException("dimension < 1");
16         _entries = new double[dimension];
17         for (int c = 0; c < _entries.length; c++) {
18             _entries[c] = 0.0D;
19         } // for (int c = 0; c < _entries.length; c++) ...
20     } // public RealRowVector(int dimension) ...
21
22     /**
23      * Erstellt eine neue RealRowVector-Instanz, die mit
24      * dem uebergebenen Array initialisiert wird.
25      * @param entries Array der Eintraege des Vektors.
26      * @exception IllegalArgumentException Wird geworfen,
27      *   wenn entries null.
28      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Wird geworfen,
29      *   wenn entries.length < 1 ist.
30      */
31     public RealRowVector(double[] entries) {
32         if (entries == null) throw new IllegalArgumentException("entries = null.");
33         int dimension = entries.length;
34         if (dimension < 1) throw new DimensionException("entries.length < 1.");
35         _entries = new double[dimension];
36         for (int c = 0; c < dimension; c++) {
37             _entries[c] = entries[c];
38         } // for (int c = 0; c < dimension; c++) ...
39     } // public RealRowVector(double[] entries) ...
40
41     /**
42      * Erstellt eine Kopie der angegebenen IRealRowVector-Instanz.
43      * @param source IRealRowVector-Instanz, die kopiert werden soll.
44      * @exception IllegalArgumentException Wird geworfen, wenn source = null ist.
45      */
46     public RealRowVector(IRealRowVector source) {
47         if (source == null) throw new IllegalArgumentException("source = null.");
48         int dimension = source.getDimension();
49         _entries = new double[dimension];
50         for (int c = 0; c < dimension; c++) {
51             _entries[c] = source.getEntry(c);
52         } // for (int c = 0; c < dimension; c++) ...
53     } // public RealRowVector(IRealRowVector source) ...
54
55     /**
56      * Gibt eine String-Repraesentation des Vektors zurueck.
57      * @return String-Repraesentation des Vektors.
58      */
59     @Override
60     public String toString() {
61         int dimension = _entries.length;
62         StringBuilder sb = new StringBuilder();
63         sb.append('(');
64         for (int c = 0; c < dimension; c++) {
65             if (c > 0) {
66                 sb.append(',');
67             } // if (c > 0) ...
68             sb.append(_entries[c]);
69         } // for (int c = 0; c < dimension; c++) ...
70         sb.append(')');
71         return sb.toString();
72     } // public String toString() ...
73
74     /**
75      * Array der Eintraege des Vektors.
76      */
77     protected double [] _entries;
78
```

```
79      /**
80       * Gibt die Dimension des Vektors zurueck.
81       * @return Dimension des Vektors.
82       */
83      public int getDimension() {
84          return _entries.length;
85      } // public int getDimension() ...
86
87      /**
88       * Gibt den Eintrag mit dem angegebenen Index zurueck.
89       * @param index Index, dessen Eintrag zurueckgegeben werden soll (Nullbasiert).
90       * @return Eintrag mit dem angegebenen Index.
91       * @exception IndexOutOfBoundsException Wird geworfen, wenn der
92       *   Index < 0 oder >= Dimension ist.
93       */
94      public double getEntry(int index) {
95          if (index < 0 || index >= _entries.length) throw new IndexOutOfBoundsException();
96          return _entries[index];
97      } // public double getEntry(int index) ...
98
99      /**
100      * Setzt den Eintrag mit dem angegebenen Index auf den angegebenen Wert.
101      * @param index Index, dessen Eintrag gesetzt werden soll (Nullbasiert).
102      * @param value Wert, auf den der Eintrag gesetzt werden soll.
103      * @exception IndexOutOfBoundsException Wird geworfen, wenn der
104      *   Index < 0 oder >= Dimension ist.
105      */
106     public void setEntry(int index, double value) {
107         if (index < 0 || index >= _entries.length) throw new IndexOutOfBoundsException();
108         _entries[index] = value;
109     } // public void setEntry(int index, double value) ...
110
111     /**
112      * Gibt eine Kopie des transponierten Vektors zurueck.
113      * @return Kopie des transponierten Vektors.
114      */
115     public IRealColumnVector transpose() {
116         return new RealColumnVector(_entries);
117     } // public IRealColumnVector transpose() ...
118
119     /**
120      * Addiert den angegebenen Wert zum Eintrag mit dem angegebenen Index.
121      * @param index Index des Eintrags.
122      * @param value Wert, der zum Eintrag addiert werden soll.
123      * @exception IndexOutOfBoundsException Wird geworfen, wenn der
124      *   Index < 0 oder >= Dimension ist.
125      */
126     public void addEntry(int index, double value) {
127          if (index < 0 || index >= _entries.length) throw new IndexOutOfBoundsException();
128          _entries[index] += value;
129     } // public void addEntry(int index, double value) ...
130
131     /**
132      * Multipliziert den Eintrag mit dem angegebenen Index mit dem angegebenen Faktor.
133      * @param index Index des Eintrags.
134      * @param factor Faktor, mit dem der Eintrag multipliziert werden soll.
135      * @exception IndexOutOfBoundsException Wird geworfen, wenn der
136      *   Index < 0 oder >= Dimension ist.
137      */
138     public void multiplyEntry(int index, double factor) {
139         if (index < 0 || index >= _entries.length) throw new IndexOutOfBoundsException();
140         _entries[index] *= factor;
141     } // public void multiplyEntry(int index, double factor) ...
142
143     /**
144      * Addiert den uebergebenen Vektor zum aktuellen Vektor.
145      * Wenn der uebergebene Vektor null ist, wird der Nullvektor addiert.
146      * @param vector Vektor, der addiert werden soll.
147      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Wird geworfen, wenn
          die Dimension
148      *   des uebergebenen Vektors nicht mit der des aktuellen Vektors uebereinstimmt.
149      */
150     public void addVector(IRealRowVector vector) {
151         if (vector != null) {
152             int dimension = _entries.length;
153             if (vector instanceof RealRowVector) {
154                 double[] vectorEntries = ((RealRowVector) vector)._entries;
155                 if (dimension != vectorEntries.length) throw new DimensionException();
156                 for (int c = 0; c < dimension; c++) {
157                     _entries[c] += vectorEntries[c];
```

```
158                 } // for (int c = 0; c < dimension; c++) ...
159             } else { // if (!(vector instanceof RealRowVector)) ...
160                 if (dimension != vector.getDimension()) throw new DimensionException();
161                 for (int c = 0; c < dimension; c++) {
162                     _entries[c] += vector.getEntry(c);
163                 } // for (int c = 0; c < dimension; c++) ...
164             } // if (!(vector instanceof RealRowVector)) ...
165         } // if (vector != null) ...
166     } // public void addVector(IRealRowVector vector) ...
167 
168     /**
169      * Addiert das factor-Fache des uebergebenen Vektors zum aktuellen Vektor.
170      * Wenn der uebergebene Vektor null ist, wird der Nullvektor addiert.
171      * @param vector Vektor, dessen factor-Faches addiert werden soll.
172      * @param factor Faktor, mit dem der uebergebene Vektor multipliziert werden soll.
173      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Wird geworfen, wenn
           die Dimension
174      *   des uebergebenen Vektors nicht mit der des aktuellen Vektors uebereinstimmt.
175      */
176     public void addVector(IRealRowVector vector, double factor) {
177         if (vector != null) {
178             int dimension = _entries.length;
179             if (vector instanceof RealRowVector) {
180                 double[] vectorEntries = ((RealRowVector) vector)._entries;
181                 if (dimension != vectorEntries.length) throw new DimensionException();
182                 for (int c = 0; c < dimension; c++) {
183                     _entries[c] += factor * vectorEntries[c];
184                 } // for (int c = 0; c < dimension; c++) ...
185             } else { // if (!(vector instanceof RealRowVector)) ...
186                 if (dimension != vector.getDimension()) throw new DimensionException();
187                 for (int c = 0; c < dimension; c++) {
188                     _entries[c] += factor * vector.getEntry(c);
189                 } // for (int c = 0; c < dimension; c++) ...
190             } // if (!(vector instanceof RealRowVector)) ...
191         } // if (vector != null) ...
192     } // public void addVector(IRealRowVector vector, double factor) ...
193 
194     /**
195      * Subtrahiert den uebergebenen Vektor vom aktuellen Vektor.
196      * Wenn der uebergebene Vektor null ist, wird der Nullvektor subtrahiert.
197      * @param vector Vektor, der subtrahiert werden soll.
198      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Wird geworfen, wenn
           die Dimension
199      *   des uebergebenen Vektors nicht mit der des aktuellen Vektors uebereinstimmt.
200      */
201     public void subtractVector(IRealRowVector vector) {
202         if (vector != null) {
203             int dimension = _entries.length;
204             if (vector instanceof RealRowVector) {
205                 double[] vectorEntries = ((RealRowVector) vector)._entries;
206                 if (dimension != vectorEntries.length) throw new DimensionException();
207                 for (int c = 0; c < dimension; c++) {
208                     _entries[c] -= vectorEntries[c];
209                 } // for (int c = 0; c < dimension; c++) ...
210             } else { // if (!(vector instanceof RealRowVector)) ...
211                 if (dimension != vector.getDimension()) throw new DimensionException();
212                 for (int c = 0; c < dimension; c++) {
213                     _entries[c] -= vector.getEntry(c);
214                 } // for (int c = 0; c < dimension; c++) ...
215             } // if (!(vector instanceof RealRowVector)) ...
216         } // if (vector != null) ...
217     } // public void subtractVector(IRealRowVector vector) ...
218 
219     /**
220      * Subtrahiert das factor-Fache des uebergebenen Vektors vom aktuellen Vektor.
221      * Wenn der uebergebene Vektor null ist, wird der Nullvektor subtrahiert.
222      * @param vector Vektor, dessen factor-Faches subtrahiert werden soll.
223      * @param factor Faktor, mit dem der uebergebene Vektor multipliziert werden soll.
224      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Wird geworfen, wenn
           die Dimension
225      *   des uebergebenen Vektors nicht mit der des aktuellen Vektors uebereinstimmt.
226      */
227     public void subtractVector(IRealRowVector vector, double factor) {
228         if (vector != null) {
229             int dimension = _entries.length;
230             if (vector instanceof RealRowVector) {
231                 double[] vectorEntries = ((RealRowVector) vector)._entries;
232                 if (dimension != vectorEntries.length) throw new DimensionException();
233                 for (int c = 0; c < dimension; c++) {
234                     _entries[c] -= factor * vectorEntries[c];
```

```
235                 } // for (int c = 0; c < dimension; c++) ...
236             } else { // if (!(vector instanceof RealRowVector)) ...
237                 if (dimension != vector.getDimension()) throw new DimensionException();
238                 for (int c = 0; c < dimension; c++) {
239                     _entries[c] -= factor * vector.getEntry(c);
240                 } // for (int c = 0; c < dimension; c++) ...
241             } // if (!(vector instanceof RealRowVector)) ...
242         } // if (vector != null) ...
243     } // public void subtractVector(IRealRowVector vector, double factor) ...
244 
245     /**
246      * Multipliziert den aktuellen Vektor mit dem uebergebenen Faktor.
247      * @param factor Faktor, mit dem der Vektor multipliziert werden soll.
248      */
249     public void scalarMultiply(double factor) {
250         int dimension = _entries.length;
251         for (int c = 0; c < dimension; c++) {
252             _entries[c] *= factor;
253         } // for (int c = 0; c < dimension; c++) ...
254     } // public void scalarMultiply(double factor) ...
255 
256     /**
257      * Ermittelt die Maximumsnorm des aktuellen Vektors.
258      * @return Maximumsnorm des aktuellen Vektors.
259      */
260     public double getNormMax() {
261         int dimension = _entries.length;
262         double max = 0.0D;
263         double currentElement;
264         // Maximum der Betraege der Koordinaten ermitteln.
265         for (int c = 0; c < dimension; c++) {
266             currentElement = _entries[c];
267             if (currentElement < 0.0D) { currentElement = -currentElement; }
268             if (currentElement > max) { max = currentElement; }
269         } // for (int c = 0; c < dimension; c++) ...
270         return max;
271     } // public double getNormMax() ...
272 
273     /**
274      * Ermittelt die l1-Norm des aktuellen Vektors.
275      * @return l1-Norm des aktuellen Vektors.
276      */
277     public double getNormSum() {
278         int dimension = _entries.length;
279         double sum = 0.0D;
280         double currentElement;
281         // Summe der Betraege der Koordinaten ermitteln.
282         for (int c = 0; c < dimension; c++) {
283             currentElement = _entries[c];
284             sum += currentElement < 0.0D ? -currentElement : currentElement;
285         } // for (int c = 0; c < dimension; c++) ...
286         return sum;
287     } // public double getNormSum() ...
288 
289     /**
290      * Ermittelt die euklidische Norm des aktuellen Vektors.
291      * @return Euklidische Norm des aktuellen Vektors.
292      */
293     public double getNorm() {
294         int dimension = _entries.length;
295         double squareSum = 0.0D;
296         double currentElement;
297         // Summe der Quadrate der Koordinaten ermitteln.
298         for (int c = 0; c < dimension; c++) {
299             currentElement = _entries[c];
300             squareSum += currentElement * currentElement;
301         } // for (int c = 0; c < dimension; c++) ...
302         return Math.sqrt(squareSum);
303     } // public double getNorm() ...
304 
305     /**
306      * Ermittelt das kanonische Skalarprodukt des aktuellen Vektors mit dem
307      * uebergebenen Vektor.
308      * @param vector Vektor, mit dem das kanonische Skalarprodukt gebildet werden soll.
309      *   Wenn vector = null ist, wird mit dem Nullvektor multipliziert.
310      * @return Kanonisches Skalarprodukt des aktuellen Vektors mit dem uebergebenen Vektor.
311      * @exception at.techmath.boltzmann.linearalgebra.DimensionException Dimension des
             uebergebenen Vektors stimmt
312      *   nicht mit der Dimension des aktuellen Vektors ueberein.
313      */
```

```
314     public double getInnerProduct(IRealRowVector vector) {
315         if (vector != null) {
316             int dimension = _entries.length;
317             if (vector instanceof RealRowVector) {
318                 double[] vectorEntries = ((RealRowVector) vector)._entries;
319                 if (dimension != _entries.length) throw new DimensionException();
320                 double result = 0.0D;
321                 for (int c = 0; c < dimension; c++) {
322                     result += _entries[c] * vectorEntries[c];
323                 } // for (int c = 0; c < dimension; c++) ...
324                 return result;
325             } else { // if (!(vector instanceof RealRowVector)) ...
326                 if (dimension != vector.getDimension()) throw new DimensionException();
327                 double result = 0.0D;
328                 for (int c = 0; c < dimension; c++) {
329                     result += _entries[c] * vector.getEntry(c);
330                 } // for (int c = 0; c < dimension; c++) ...
331                 return result;
332             } // if (!(vector instanceof RealRowVector)) ...
333         } else { // if (vector == null) ...
334             return 0.0D;
335         } // if (vector == null) ...
336     } // public double getInnerProduct(IRealRowVector vector) ...
337 } // public class RealRowVector implements IMutableRealRowVector ...
```

Listing 26: Klasse *at.techmath.boltzmann.linearalgebra.SingularException*

```
1  package at.techmath.boltzmann.linearalgebra;
2
3  /**
4   * Exception, die geworfen wird, wenn eine singulaere Matrix verwendet wird,
5   * wo eine regulaere Matrix erwartet wird.
6   * @author Stefan Schnabl (e0226245)
7   */
8  public class SingularException extends RuntimeException {
9      private static final long serialVersionUID = 52407034007763643 40L;
10     public SingularException() { super(); }
11     public SingularException(String message) { super(message); }
12     public SingularException(String message, Throwable cause) { super(message, cause); }
13     public SingularException(Throwable cause) { super(cause); }
14 } // public class SingularException extends RuntimeException ...
```

A.1.3 Package at.techmath.boltzmann.oracle

Listing 27: Klasse *at.techmath.boltzmann.oracle.BoltzmannOracleException*

```
1  package at.techmath.boltzmann.oracle;
2
3  /**
4   * Exception, die geworfen wird, falls
5   * das Boltzmann-Orakel nicht auswerten kann.
6   * @author Stefan Schnabl (e0226245)
7   */
8  public class BoltzmannOracleException extends RuntimeException {
9      private static final long serialVersionUID = -1613848718538556263L;
10     public BoltzmannOracleException() { super(); }
11     public BoltzmannOracleException(String message) { super(message); }
12     public BoltzmannOracleException(String message, Throwable cause) { super(message, cause); }
13     public BoltzmannOracleException(Throwable cause) { super(cause); }
14 } // public class BoltzmannOracleException extends RuntimeException ...
```

Listing 28: Klasse *at.techmath.boltzmann.oracle.DefaultExponentialBoltzmannOracleFactory*

```
1  package at.techmath.boltzmann.oracle;
2
3  import at.techmath.boltzmann.specification.ISpecification;
4  import at.techmath.boltzmann.linearalgebra.DefaultAlgorithms;
5
6  /**
7   * Standard-Factory fuer exponentielle Boltzmann-Orakel.
8   * @author Stefan Schnabl (e0226245).
9   */
10 public class DefaultExponentialBoltzmannOracleFactory implements
       IExponentialBoltzmannOracleFactory {
11     /**
12      * Erzeugt fuer die angegebene Spezifikation
13      * ein exponentielles Boltzmann-Orakel.
14      * @param specification Spezifikation, fuer die das Orakel erzeugt werden soll.
15      * @return Exponentielles Boltzmann-Orakel fuer die angegebene Spezifikation.
16      * @throws IllegalArgumentException Wird geworfen, wenn die
17      *   uebergebene Spezifikation null ist.
18      */
19     public IExponentialBoltzmannOracle createOracle(ISpecification specification) {
20         if (specification == null) {
21             throw new IllegalArgumentException("specification = null.");
22         } // if (specification == null) ...
23         ExponentialBoltzmannOracle result = new ExponentialBoltzmannOracle(specification,
               DefaultAlgorithms.getDefaultRealLinearSolverAlgorithm());
24         return result;
25     } // public IExponentialBoltzmannOracle createOracle(ISpecification specification) ...
26 } // public class DefaultExponentialBoltzmannOracleFactory implements
       IExponentialBoltzmannOracleFactory ...
```

Listing 29: Klasse *at.techmath.boltzmann.oracle.ExponentialBoltzmannOracle*

```
1  package at.techmath.boltzmann.oracle;
2
3  import at.techmath.boltzmann.specification.*;
4  import at.techmath.boltzmann.linearalgebra.*;
5
6  import java.util.TreeMap;
7
8  /**
9   * Standard-Implementierung fuer das exponentielle Boltzmann-Orakel.
10  * @author Stefan Schnabl (e0226245)
11  */
12 class ExponentialBoltzmannOracle implements IExponentialBoltzmannOracle {
13     /**
14      * Erzeugt eine neue Instanz.
15      * @param specification Kombinatorische Spezifikation, fuer die das Orakel
16      *   erzeugt werden soll.
17      * @param solver Loeser fuer lineare Gleichungssysteme, der verwendet werden soll.
18      * @throws IllegalArgumentException Wird geworfen, wenn mindestens
19      *   eines der Argumente null ist.
20      */
21     public ExponentialBoltzmannOracle(ISpecification specification,
22         IRealLinearSolverAlgorithm solver) {
23         super();
24         if (specification == null) throw new IllegalArgumentException("specification = null.");
25         if (solver == null) throw new IllegalArgumentException("solver = null.");
26         _specification = specification;
27         _solver = solver;
28         _isBuilt = false;
29         _cachedValueMap = new TreeMap<Double, IRealVector>();
30         buildStructure();
31     } // public ExponentialBoltzmannOracle(ISpecification specification, ...) ...
32
33     /**
34      * Wurde das Orakel aufgebaut?
35      */
36     private boolean _isBuilt;
37
38     /**
39      * Baut die Orakel-Komponenten auf.
40      * Darf nur im Konstruktor aufgerufen werden.
41      */
42     private void buildStructure() {
43         if (_isBuilt) {
44             throw new IllegalStateException("buildStructure darf nur im Konstruktor aufgerufen
                    werden.");
45         } // if (isBuilt) ...
46         _isBuilt = true;
47         _speciesCount = _specification.getCount();
48         _exponentialGeneratingFunctions = new ExponentialGeneratingFunction[_speciesCount];
49         for (int c = 0; c < _speciesCount; c++) {
50             INamedSpecies currentSpecies = _specification.getSpeciesIndex(c);
51             ExponentialGeneratingFunction currentGeneratingFunction = null;
52             if (currentSpecies instanceof ISpeciesSum) {
53                 // Summen-Spezies
54                 ISpeciesSum currentSpeciesSum = (ISpeciesSum) currentSpecies;
55                 ExponentialGeneratingFunctionSum egfSum = new ExponentialGeneratingFunctionSum(
                        _speciesCount);
56                 currentGeneratingFunction = egfSum;
57                 int addendCount = currentSpeciesSum.getAddendCount();
58                 for (int currentAddendIndex = 0; currentAddendIndex < addendCount;
                        currentAddendIndex++) {
59                     ISpeciesReference currentReference = currentSpeciesSum.getAddendIndex(
                            currentAddendIndex);
60                     ISpecies referencedSpecies = currentReference.getSpecies();
61                     if (referencedSpecies instanceof ISpeciesOne) {
62                         egfSum.incrementSpeciesOneCount();
63                     } else if (referencedSpecies instanceof ISpeciesSingleton) {
64                         egfSum.incrementSpeciesSingletonCount();
65                     } else if (referencedSpecies instanceof INamedSpecies) {
66                         egfSum.incrementSpeciesIndexCount(((INamedSpecies) referencedSpecies).
                                getIndex());
67                     }
68                 } // for (int currentAddendIndex = 0; currentAddendIndex < addendCount;
                        currentAddendIndex++) ...
69             } else if (currentSpecies instanceof ISpeciesProduct) {
70                 // Produkt-Spezies
71                 ISpeciesProduct currentSpeciesProduct = (ISpeciesProduct) currentSpecies;
72                 ExponentialGeneratingFunctionProduct egfProduct = new
                        ExponentialGeneratingFunctionProduct(_speciesCount);
```

```
73                  currentGeneratingFunction = egfProduct;
74                  int factorCount = currentSpeciesProduct.getFactorCount();
75                  for (int currentFactorIndex = 0; currentFactorIndex < factorCount;
                        currentFactorIndex++) {
76                      ISpeciesReference currentReference = currentSpeciesProduct.getFactorIndex(
                            currentFactorIndex);
77                      ISpecies referencedSpecies = currentReference.getSpecies();
78                      if (referencedSpecies instanceof ISpeciesSingleton) {
79                          egfProduct.incrementSpeciesSingletonCount();
80                      } else if (referencedSpecies instanceof INamedSpecies) {
81                          egfProduct.incrementSpeciesIndexCount(((INamedSpecies)
                                referencedSpecies).getIndex());
82                      }
83                  } // for (int currentFactorIndex = 0; currentFactorIndex < factorCount;
                        currentFactorIndex++) ...
84              } else if (currentSpecies instanceof ISpeciesSequence) {
85                  // Sequence-Spezies
86                  ISpeciesSequence currentSpeciesSequence = (ISpeciesSequence) currentSpecies;
87                  ISpeciesReference currentSpeciesReference = currentSpeciesSequence.
                        getSpeciesParameter();
88                  ISpecies referencedSpecies = currentSpeciesReference.getSpecies();
89                  if (referencedSpecies instanceof ISpeciesSingleton) {
90                      currentGeneratingFunction = new
                            ExponentialGeneratingFunctionSequenceSingleton(_speciesCount);
91                  } else if (referencedSpecies instanceof INamedSpecies) {
92                      currentGeneratingFunction = new ExponentialGeneratingFunctionSequence(
93                          ((INamedSpecies) referencedSpecies).getIndex(), _speciesCount);
94                  }
95              } else if (currentSpecies instanceof ISpeciesSet) {
96                  // Set-Spezies
97                  ISpeciesSet currentSpeciesSet = (ISpeciesSet) currentSpecies;
98                  ISpeciesReference currentSpeciesReference = currentSpeciesSet.
                        getSpeciesParameter();
99                  ISpecies referencedSpecies = currentSpeciesReference.getSpecies();
100                 if (referencedSpecies instanceof ISpeciesSingleton) {
101                     currentGeneratingFunction = new ExponentialGeneratingFunctionSetSingleton(
                            _speciesCount);
102                 } else if (referencedSpecies instanceof INamedSpecies) {
103                     currentGeneratingFunction = new ExponentialGeneratingFunctionSet(
104                         ((INamedSpecies) referencedSpecies).getIndex(), _speciesCount);
105                 }
106             } else if (currentSpecies instanceof ISpeciesCycle) {
107                 // Cycle-Spezies
108                 ISpeciesCycle currentSpeciesCycle = (ISpeciesCycle) currentSpecies;
109                 ISpeciesReference currentSpeciesReference = currentSpeciesCycle.
                        getSpeciesParameter();
110                 ISpecies referencedSpecies = currentSpeciesReference.getSpecies();
111                 if (referencedSpecies instanceof ISpeciesSingleton) {
112                     currentGeneratingFunction = new ExponentialGeneratingFunctionCycleSingleton
                            (_speciesCount);
113                 } else if (referencedSpecies instanceof INamedSpecies) {
114                     currentGeneratingFunction = new ExponentialGeneratingFunctionCycle(
115                         ((INamedSpecies) referencedSpecies).getIndex(), _speciesCount);
116                 }
117             }
118             if (currentGeneratingFunction == null) {
119                 throw new IllegalArgumentException("Spezifikation nicht kompatibel.");
120             } // if (currentGeneratingFunction == null) ..
121             _exponentialGeneratingFunctions[c] = currentGeneratingFunction;
122         } // for (int c = 0; c < _speciesCount; c++) ...
123     } // protected void buildStructure() ...
124
125     /**
126      * Anzahl der benannten Spezies.
127      */
128     private int _speciesCount = 0;
129
130     /**
131      * Exponentielle erzeugende Funktionen des Systems, indiziert mit
132      * dem Index der jeweiligen Spezies der Spezifikation.
133      */
134     private ExponentialGeneratingFunction[] _exponentialGeneratingFunctions = null;
135
136     /**
137      * Abbildung Parameter -> Wert der erzeugenden Funktionen.
138      */
139     private TreeMap<Double, IRealVector> _cachedValueMap;
140
141     /**
142      * Wenn H(Z,Y) das kombinatorische System bezeichnet, dann ermittelt
```

```
143      * diese Methode H (Z,Y) (x, currentVector).
144      * @param currentVector Aktueller Wert der erzeugenden Funktionen der System-Spezies.
145      * @param x Parameter, an dem die erzeugende Funktion ausgewertet wird.
146      * @param resultVector Vector, in den das Ergebnis ausgegeben wird.
147      * @throws IllegalArgumentException Wird geworfen, wenn die Vektoren null
148      *   sind oder die falsche Dimension besitzen.
149      */
150     private void evaluateSystem(RealColumnVector currentVector, double x, RealColumnVector
             resultVector) {
151         int speciesCount = _speciesCount;
152         if (currentVector == null) {
153             throw new IllegalArgumentException("currentVector = null.");
154         } // if (currentVector == null) ...
155         if (resultVector == null) {
156             throw new IllegalArgumentException("resultVector = null.");
157         } // if (resultVector == null) ...
158         if (currentVector.getDimension() != speciesCount) {
159             throw new IllegalArgumentException("currentVector hat falsche Dimension.");
160         } // if (currentVector.getDimension() != speciesCount) ...
161         if (resultVector.getDimension() != speciesCount) {
162             throw new IllegalArgumentException("resultVector hat falsche Dimension.");
163         } // if (resultVector.getDimension() != speciesCount) ...
164         for (int c = 0; c < speciesCount; c++) {
165             resultVector.setEntry(c, _exponentialGeneratingFunctions[c].evaluate(x,
                     currentVector));
166         } // for (int c = 0; c < speciesCount; c++) ...
167     } // private void evaluateSystem(RealColumnVector currentVector, double x, RealColumnVector
             resultVector) ...
168
169     /**
170      * Wenn H(Z,Y) das kombinatorische System bezeichnet und DH/DY(Z,Y) die Jacobi-Matrix,
171      * dann ermittelt diese Methode (Id - DH/DY(Z,Y))(x, currentVector).
172      * @param currentVector Aktueller Wert der erzeugenden Funktionen der System-Spezies.
173      * @param x Parameter, an dem die erzeugende Funktion ausgewertet wird.
174      * @param resultMatrix Matrix, in die das Ergebnis ausgegeben wird.
175      * @throws IllegalArgumentException Wird geworfen, wenn der Vektor oder
176      *   die Matrix null ist bzw. die falsche Dimension besitzt.
177      */
178     private void evaluateIdentityMinusJacobian(RealColumnVector currentVector, double x,
             RealMatrix resultMatrix) {
179         int speciesCount = _speciesCount;
180         if (currentVector == null) {
181             throw new IllegalArgumentException("currentVector = null.");
182         } // if (currentVector == null) ...
183         if (resultMatrix == null) {
184             throw new IllegalArgumentException("resultMatrix = null.");
185         } // if (resultMatrix == null) ...
186         if (currentVector.getDimension() != speciesCount) {
187             throw new IllegalArgumentException("currentVector hat falsche Dimension.");
188         } // if (currentVector.getDimension() != speciesCount) ...
189         if (resultMatrix.getRowCount() != _speciesCount || resultMatrix.getColumnCount() !=
                 _speciesCount) {
190             throw new IllegalArgumentException("resultMatrix hat falsche Dimension.");
191         } // if (resultMatrix.getRowCount() != _speciesCount || resultMatrix.getColumnCount()
                 != _speciesCount) ...
192         for (int currentRow = 0; currentRow < speciesCount; currentRow++) {
193             ExponentialGeneratingFunction currentEgf = _exponentialGeneratingFunctions[
                     currentRow];
194             for (int currentColumn = 0; currentColumn < speciesCount; currentColumn++) {
195                 double newValue = (currentRow == currentColumn ? 1.0D : 0.0D) -
196                     currentEgf.evaluatePartialDerivative(currentColumn, x, currentVector);
197                 resultMatrix.setEntry(currentRow, currentColumn, newValue);
198             } // for (int currentColumn = 0; currentColumn < speciesCount; currentColumn++) ...
199         } // for (int currentRow = 0; currentRow < speciesCount; currentRow++) ...
200     } // private void evaluateIdentityMinusJacobian(RealColumnVector currentVector, double x,
             RealMatrix resultMatrix) ...
201
202     /**
203      * Wertet die erzeugende Funktion der Spezifikation
204      * des Orakels fuer die Spezies mit dem angegebenen
205      * Index fuer den angegebenen Parameter aus.
206      * @param index Index der Spezies innerhalb der Spezifikation.
207      * @param x Parameter der exponentiellen erzeugenden Funktion.
208      * @return Wert der erzeugenden Funktion der angegebenen
209      *   Spezies an der angegebenen Stelle.
210      */
211     public double evaluate(int index, double x) {
212         if (_cachedValueMap.containsKey(x)) {
213             IRealVector vector = _cachedValueMap.get(x);
214             if (vector != null) {
```

```
215                 return vector.getEntry(index);
216             } else { // if (vector == null) ...
217                 throw new BoltzmannOracleException("Fehler beim Auswerten der erzeugenden
                        Funktionen.");
218             } // if (vector == null) ...
219         } else {
220             // Newton-Iteration
221             // Paper: Algorithms for Combinatorial Structures - Well-Founded Systems and Newton
                    Iterations.
222             // Seite 43, Satz 9.13
223             // y[0] = 0
224             // y[n+1] = y[n] + (Id - DH/DY(x, y[n])) * (H(x, y[n]) - y[n])
225             //
226             int speciesCount = _speciesCount;
227             RealColumnVector vectorH = new RealColumnVector(speciesCount);
228             RealColumnVector vectorY = new RealColumnVector(speciesCount);
229             RealMatrix matrixIdMinusDH = new RealMatrix(speciesCount, speciesCount);
230             //System.out.println("Y: " + vectorY.toString());
231             for (int c = 0; c < 40; c++) {
232                 //System.out.println("Schritt " + (c + 1));
233                 evaluateSystem(vectorY, x, vectorH);
234                 evaluateIdentityMinusJacobian(vectorY, x, matrixIdMinusDH);
235                 //System.out.println("H: " + vectorH.toString());
236                 //System.out.println("DH: " + matrixIdMinusDH.toString());
237                 vectorH.subtractVector(vectorY);
238                 IRealColumnVector vectorSolution = _solver.solve(matrixIdMinusDH, vectorH);
239                 vectorY.addVector(vectorSolution);
240             } // for (int c = 0; c < 40; c++) ...
241             //System.out.println("Y: " + vectorY.toString());
242             _cachedValueMap.put(x, vectorY);
243             return vectorY.getEntry(index);
244         }
245     } // public double evaluate(int index, double x) ...
246
247     /**
248      * Spezifikation, anhand der das Boltzmann-Orakel erzeugt werden soll.
249      */
250     private ISpecification _specification;
251
252     /**
253      * Gibt die Spezifikation des Orakels zurueck.
254      * @return Spezifikation des Orakels.
255      */
256     public ISpecification getSpecification() {
257         return _specification;
258     } // public ISpecification getSpecification() ...
259
260     /**
261      * Algorithmus zur Loesung von linearen Gleichungssystemen.
262      */
263     private IRealLinearSolverAlgorithm _solver;
264
265     /**
266      * Gibt eine String-Repraesentation der Instanz zurueck.
267      * @return String-Repraesentation der Instanz.
268      */
269     @Override
270     public String toString() {
271         StringBuilder sb = new StringBuilder();
272         sb.append("Exponential-Oracle\n");
273         for (int c = 0; c < _exponentialGeneratingFunctions.length; c++) {
274             sb.append(_exponentialGeneratingFunctions[c].toString());
275         } // for (int c = 0; c < _exponentialGeneratingFunctions.length; c++) ...
276         return sb.toString();
277     } // public String toString() ...
278 } // class ExponentialBoltzmannOracle implements IExponentialBoltzmannOracle ...
```

Listing 30: Klasse *at.techmath.boltzmann.oracle.ExponentialGeneratingFunction*

```
1  package at.techmath.boltzmann.oracle;
2
3  import at.techmath.boltzmann.linearalgebra.IRealVector;
4
5  /**
6   * Basisklasse fuer die exponentiellen erzeugenden
7   * Funktionen der verschiedenen Spezies.
8   * @author Stefan Schnabl (e0226245)
9   */
10 abstract class ExponentialGeneratingFunction {
11     /**
12      * Meldung der Exception, die anzeigt, dass
13      * der angegebene Spezies-Index ausserhalb des gueltigen
14      * Bereichs ist.
15      */
16     protected static final String EXCEPTION_MESSAGE_SPECIES_INDEX = "Angegebener Spezies-Index
           ausserhalb des gueltigen Bereichs.";
17
18     /**
19      * Meldung der Exception, die anzeigt, dass
20      * der angegebene System-Vektor ungueltig ist.
21      */
22     protected static final String EXCEPTION_MESSAGE_SYSTEM_VECTOR_INVALID = "Angegebener System
           -Vektor ungueltig.";
23
24     /**
25      * Wertet die exponentielle erzeugende Funktion an der
26      * uebergebenen Stelle aus.
27      * @param x Stelle, an der die EEF ausgewertet werden soll.
28      * @param systemValues Vektor der Werte der exponentiellen
29      *   erzeugenden Funktionen des Systems. (Index entspricht dem
30      *   Index der Spezies in der Spezifikation)
31      * @return Wert der exponentiellen erzeugenden Funktion an
32      *   der angegebenen Stelle.
33      * @throws IllegalArgumentException Wird geworfen,
34      *   wenn der angegebene Vektor null ist, nicht die richtige
35      *   Dimension besitzt, oder der Parameter
36      *   ausserhalb des Definitionsbereichs der erzeugenden
37      *   Funktion liegt.
38      */
39     public abstract double evaluate(double x, IRealVector systemValues);
40
41     /**
42      * Wertet die partiell nach der Spezies mit dem angegebenen
43      * Index abgeleitete erzeugende Funktion an der angegebenen
44      * Stelle aus.
45      * @param index Index der Spezies, nach der abgeleitet werden soll.
46      * @param x Wert, an dem die erzeugende Funktion ausgewertet werden soll.
47      * @param systemValues Vektor der Werte der exponentiellen
48      *   erzeugenden Funktionen des Systems. (Index entspricht dem
49      *   Index der Spezies in der Spezifikation)
50      * @return Wert der partiellen Ableitung an der angegebenen Stelle.
51      * @throws IllegalArgumentException Wird geworfen,
52      *   wenn der angegebene Vektor null ist, nicht die richtige Dimension
53      *   besitzt, oder der Parameter
54      *   ausserhalb des Definitionsbereichs der erzeugenden
55      *   Funktion liegt.
56      * @throws IndexOutOfBoundsException wird geworfen,
57      *   wenn der angegebene Index ausserhalb des gueltigen Bereichs liegt.
58      */
59     public abstract double evaluatePartialDerivative(int index, double x, IRealVector
           systemValues);
60 } // abstract class ExponentialGeneratingFunction ...
```

Listing 31: Klasse *at.techmath.boltzmann.oracle.ExponentialGeneratingFunctionCycle*

```
1  package at.techmath.boltzmann.oracle;
2
3  import at.techmath.boltzmann.linearalgebra.IRealVector;
4
5  /**
6   * Exponentielle erzeugende Funktion der Cycle-Species,
7   * in die eine benannte Spezies eingesetzt wird.
8   * @author Stefan Schnabl (e0226245)
9   */
10 class ExponentialGeneratingFunctionCycle extends ExponentialGeneratingFunction {
11     /**
12      * Meldung der Exception, die anzeigt,
13      * dass der Parameter ausserhalb des Konvergenzradiuses liegt.
14      */
15     protected static final String EXCEPTION_MESSAGE_RADIUS_OF_CONVERGENCE = "Parameter
           ausserhalb des Konvergenzradius der Cycle-EEF.";
16
17     /**
18      * Erzeugt eine neue Instanz.
19      * @param speciesIndex Index der Spezies.
20      * @param speciesCount Anzahl der Spezies.
21      */
22     public ExponentialGeneratingFunctionCycle(int speciesIndex, int speciesCount) {
23         super();
24         _speciesIndex = speciesIndex;
25         _speciesCount = speciesCount;
26     } // public ExponentialGeneratingFunctionCycle(int speciesIndex, int speciesCount) ...
27
28     /**
29      * Index der Spezies, die eingesetzt wird.
30      */
31     private int _speciesIndex;
32
33     /**
34      * Anzahl der Spezies.
35      */
36     private int _speciesCount;
37
38     /**
39      * Wertet die exponentielle erzeugende Funktion an der
40      * uebergebenen Stelle aus.
41      * @param x Stelle, an der die EEF ausgewertet werden soll.
42      * @param systemValues Vektor der Werte der exponentiellen
43      *   erzeugenden Funktionen des Systems. (Index entspricht dem
44      *   Index der Spezies in der Spezifikation)
45      * @return Wert der exponentiellen erzeugenden Funktion an
46      *   der angegebenen Stelle.
47      * @throws IllegalArgumentException Wird geworfen,
48      *   wenn der angegebene Vektor null ist, nicht die richtige
49      *   Dimension besitzt, oder der Parameter
50      *   ausserhalb des Definitionsbereichs der erzeugenden
51      *   Funktion liegt.
52      */
53     @Override
54     public double evaluate(double x, IRealVector systemValues) {
55         if (systemValues == null || systemValues.getDimension() != _speciesCount) {
56             throw new IllegalArgumentException(EXCEPTION_MESSAGE_SYSTEM_VECTOR_INVALID);
57         } // if (systemValues == null || systemValues.getDimension() != _speciesCount) ...
58         double value = systemValues.getEntry(_speciesIndex);
59         if (value < 0.0D || value >= 1.0D) {
60             throw new IllegalArgumentException(EXCEPTION_MESSAGE_RADIUS_OF_CONVERGENCE);
61         } // if (value < 0.0D || value >= 1.0D) ...
62         return Math.log(1.0D / (1.0D - value));
63     } // public double evaluate(double x, IRealVector systemValues) ...
64
65     /**
66      * Wertet die partiell nach der Spezies mit dem angegebenen
67      * Index abgeleitete erzeugende Funktion an der angegebenen
68      * Stelle aus.
69      * @param index Index der Spezies, nach der abgeleitet werden soll.
70      * @param x Wert, an dem die erzeugende Funktion ausgewertet werden soll.
71      * @param systemValues Vektor der Werte der exponentiellen
72      *   erzeugenden Funktionen des Systems. (Index entspricht dem
73      *   Index der Spezies in der Spezifikation)
74      * @return Wert der partiellen Ableitung an der angegebenen Stelle.
75      * @throws IllegalArgumentException Wird geworfen,
76      *   wenn der angegebene Vektor null ist, nicht die richtige
77      *   Dimension besitzt, oder der Parameter
78      *   ausserhalb des Definitionsbereichs der erzeugenden
```

```
79       *   Funktion liegt.
80       * @throws IndexOutOfBoundsException wird geworfen,
81       *   wenn der angegebene Index ausserhalb des gueltigen Bereichs liegt.
82       */
83      @Override
84      public double evaluatePartialDerivative(int index, double x, IRealVector systemValues) {
85          if (index < 0 || index >= _speciesCount) {
86              throw new IndexOutOfBoundsException(EXCEPTION_MESSAGE_SPECIES_INDEX);
87          } // if (index < 0 || index >= _speciesCount) ...
88          if (systemValues == null || systemValues.getDimension() != _speciesCount) {
89              throw new IllegalArgumentException(EXCEPTION_MESSAGE_SYSTEM_VECTOR_INVALID);
90          } // if (systemValues == null || systemValues.getDimension() != _speciesCount) ...
91          if (index == _speciesIndex) {
92              double value = systemValues.getEntry(_speciesIndex);
93              if (value < 0.0D || value >= 1.0D) {
94                  throw new IllegalArgumentException(EXCEPTION_MESSAGE_RADIUS_OF_CONVERGENCE);
95              } // if (value < 0.0D || value >= 1.0D) ...
96              return 1.0D / (1.0D - value);
97          } else { // if (index != _speciesIndex) ...
98              return 0.0D;
99          } // if (index != _speciesIndex) ...
100     } // public double evaluatePartialDerivative(int index, double x, IRealVector systemValues)
          ...
101
102     /**
103      * Gibt eine String-Repraesentation der Instanz zurueck.
104      * @return String-Repraesentation der Instanz.
105      */
106     @Override
107     public String toString() {
108         StringBuilder sb = new StringBuilder();
109         sb.append("  Cycle ");
110         sb.append("Y_").append(_speciesIndex);
111         sb.append("\n");
112         return sb.toString();
113     } // public String toString() ...
114 } // class ExponentialGeneratingFunctionSet extends ExponentialGeneratingFunction ...
```

Listing 32: Klasse *at.techmath.boltzmann.oracle.ExponentialGeneratingFunctionCycleSingleton*

```
1  package at.techmath.boltzmann.oracle;
2
3  import at.techmath.boltzmann.linearalgebra.IRealVector;
4
5  /**
6   * Exponentielle erzeugende Funktion der Cycle-Species,
7   * in die die Singleton-Spezies eingesetzt wird.
8   * @author Stefan Schnabl (e0226245)
9   */
10 class ExponentialGeneratingFunctionCycleSingleton extends ExponentialGeneratingFunction {
11     /**
12      * Meldung der Exception, die anzeigt,
13      * dass der Parameter ausserhalb des Konvergenzradiuses liegt.
14      */
15     protected static final String EXCEPTION_MESSAGE_RADIUS_OF_CONVERGENCE = "Parameter
           ausserhalb des Konvergenzradius der Cycle-EEF.";
16
17     /**
18      * Erzeugt eine neue Instanz.
19      * @param speciesCount Anzahl der Spezies.
20      */
21     public ExponentialGeneratingFunctionCycleSingleton(int speciesCount) {
22         super();
23         _speciesCount = speciesCount;
24     } // public ExponentialGeneratingFunctionCycleSingleton(int speciesCount) ...
25
26     /**
27      * Anzahl der Spezies.
28      */
29     private int _speciesCount;
30
31     /**
32      * Wertet die exponentielle erzeugende Funktion an der
33      * uebergebenen Stelle aus.
34      * @param x Stelle, an der die EEF ausgewertet werden soll.
35      * @param systemValues Vektor der Werte der exponentiellen
36      *   erzeugenden Funktionen des Systems. (Index entspricht dem
37      *   Index der Spezies in der Spezifikation)
38      * @return Wert der exponentiellen erzeugenden Funktion an
39      *   der angegebenen Stelle.
40      * @throws IllegalArgumentException Wird geworfen,
41      *   wenn der angegebene Vektor null ist, nicht die richtige
42      *   Dimension besitzt, oder der Parameter
43      *   ausserhalb des Definitionsbereichs der erzeugenden
44      *   Funktion liegt.
45      */
46     @Override
47     public double evaluate(double x, IRealVector systemValues) {
48         if (systemValues == null || systemValues.getDimension() != _speciesCount) {
49             throw new IllegalArgumentException(EXCEPTION_MESSAGE_SYSTEM_VECTOR_INVALID);
50         } // if (systemValues == null || systemValues.getDimension() != _speciesCount) ...
51         if (x < 0.0D || x >= 1.0D) {
52             throw new IllegalArgumentException(EXCEPTION_MESSAGE_RADIUS_OF_CONVERGENCE);
53         } // if (x < 0.0D || x >= 1.0D) ...
54         return Math.log(1.0D / (1.0D - x));
55     } // public double evaluate(double x, IRealVector systemValues) ...
56
57     /**
58      * Wertet die partiell nach der Spezies mit dem angegebenen
59      * Index abgeleitete erzeugende Funktion an der angegebenen
60      * Stelle aus.
61      * @param index Index der Spezies, nach der abgeleitet werden soll.
62      * @param x Wert, an dem die erzeugende Funktion ausgewertet werden soll.
63      * @param systemValues Vektor der Werte der exponentiellen
64      *   erzeugenden Funktionen des Systems. (Index entspricht dem
65      *   Index der Spezies in der Spezifikation)
66      * @return Wert der partiellen Ableitung an der angegebenen Stelle.
67      * @throws IllegalArgumentException Wird geworfen,
68      *   wenn der angegebene Vektor null ist, nicht die richtige
69      *   Dimension besitzt, oder der Parameter
70      *   ausserhalb des Definitionsbereichs der erzeugenden
71      *   Funktion liegt.
72      * @throws IndexOutOfBoundsException wird geworfen,
73      *   wenn der angegebene Index ausserhalb des gueltigen Bereichs liegt.
74      */
75     @Override
76     public double evaluatePartialDerivative(int index, double x, IRealVector systemValues) {
77         if (index < 0 || index >= _speciesCount) {
78             throw new IndexOutOfBoundsException(EXCEPTION_MESSAGE_SPECIES_INDEX);
```

```
79          } // if (index < 0 || index >= _speciesCount) ...
80          if (systemValues == null || systemValues.getDimension() != _speciesCount) {
81              throw new IllegalArgumentException(EXCEPTION_MESSAGE_SYSTEM_VECTOR_INVALID);
82          } // if (systemValues == null || systemValues.getDimension() != _speciesCount) ...
83          return 0.0D;
84      } // public double evaluatePartialDerivative(int index, double x, IRealVector systemValues)
            ...
85
86      /**
87       * Gibt eine String-Repraesentation der Instanz zurueck.
88       * @return String-Repraesentation der Instanz.
89       */
90      @Override
91      public String toString() {
92          StringBuilder sb = new StringBuilder();
93          sb.append("  Cycle ");
94          sb.append("Z");
95          sb.append("\n");
96          return sb.toString();
97      } // public String toString() ...
98  } // class ExponentialGeneratingFunctionCycleSingleton extends ExponentialGeneratingFunction
        ...
```

Listing 33: Klasse *at.techmath.boltzmann.oracle.ExponentialGeneratingFunctionProduct*

```
1  package at.techmath.boltzmann.oracle;
2
3  import at.techmath.boltzmann.linearalgebra.IRealVector;
4
5  /**
6   * Exponentielle erzeugende Funktion der Produkt-Spezies.
7   * @author Stefan Schnabl (e0226245)
8   */
9  class ExponentialGeneratingFunctionProduct extends ExponentialGeneratingFunction {
10     /**
11      * Erzeugt eine neue Instanz.
12      * @param speciesCount Anzahl der Spezies.
13      * @throws IllegalArgumentException Wird geworfen, wenn der speciesCount
14      *   negativ ist.
15      */
16     public ExponentialGeneratingFunctionProduct(int speciesCount) {
17         super();
18         if (speciesCount < 0) {
19             throw new IllegalArgumentException("speciesCount < 0.");
20         } // if (speciesCount < 0) ...
21         this._speciesCount = speciesCount;
22         this._speciesSingletonCount = 0;
23         int[] speciesIndexCount = this._speciesIndexCount = new int[speciesCount];
24         for (int c = 0; c < speciesCount; c++) {
25             speciesIndexCount[c] = 0;
26         } // for (int c = 0; c < speciesCount; c++) ...
27     } // public ExponentialGeneratingFunctionProduct(int speciesCount) ...
28
29     /**
30      * Anzahl der Spezies.
31      */
32     private int _speciesCount;
33
34     /**
35      * Anzahl der Summanden, die die Singleton-Spezies sind.
36      */
37     private int _speciesSingletonCount;
38
39     /**
40      * Erhoeht die Anzahl der Summanden, die die Singleton-Spezies sind, um eins.
41      */
42     protected void incrementSpeciesSingletonCount() {
43         _speciesSingletonCount += 1;
44     } // protected void incrementSpeciesSingletonCount() ...
45
46     /**
47      * Anzahl der Summanden, die benannte Spezies sind.
48      * Der Index ins Array entspricht dem Index der benannten Spezies.
49      */
50     private int[] _speciesIndexCount;
51
52     /**
53      * Erhoeht die Anzahl der Summanden, die die benannte Spezies mit
54      * dem angegebenen Index sind, um eins.
55      * @param index Index der Spezies.
56      * @throws IndexOutOfBoundsException Wird geworfen, wenn der Index
57      *   ausserhalb des gueltigen Bereichs liegt.
58      */
59     protected void incrementSpeciesIndexCount(int index) {
60         if (index < 0 || index >= _speciesCount) {
61             throw new IndexOutOfBoundsException("Index ausserhalb des gueltigen Bereichs.");
62         } // if (index < 0 || index >= _speciesCount) ...
63         _speciesIndexCount[index] += 1;
64     } // protected void incrementSpeciesIndexCount(int index) ...
65
66     /**
67      * Wertet die exponentielle erzeugende Funktion an der
68      * uebergebenen Stelle aus.
69      * @param x Stelle, an der die EEF ausgewertet werden soll.
70      * @param systemValues Vektor der Werte der exponentiellen
71      *   erzeugenden Funktionen des Systems. (Index entspricht dem
72      *   Index der Spezies in der Spezifikation)
73      * @return Wert der exponentiellen erzeugenden Funktion an
74      *   der angegebenen Stelle.
75      * @throws IllegalArgumentException Wird geworfen,
76      *   wenn der angegebene Vektor null ist, nicht die richtige
77      *   Dimension besitzt, oder der Parameter
78      *   ausserhalb des Definitionsbereichs der erzeugenden
79      *   Funktion liegt.
```

```
80      */
81     @Override
82     public double evaluate(double x, IRealVector systemValues) {
83         if (systemValues == null || systemValues.getDimension() != _speciesCount) {
84             throw new IllegalArgumentException(EXCEPTION_MESSAGE_SYSTEM_VECTOR_INVALID);
85         } // if (systemValues == null || systemValues.getDimension() != _speciesCount) ...
86         double result = 1.0D;
87         // Singleton-Spezies behandeln
88         int currentExponent = _speciesSingletonCount;
89         if (currentExponent > 0) {
90             result *= Math.pow(x, currentExponent);
91         } // if (currentExponent > 0) ...
92         int speciesCount = _speciesCount;
93         int[] speciesIndexCount = _speciesIndexCount;
94         for (int c = 0; c < speciesCount; c++) {
95             currentExponent = speciesIndexCount[c];
96             if (currentExponent > 0) {
97                 result *= Math.pow(systemValues.getEntry(c), currentExponent);
98             } // if (currentExponent > 0) ...
99         } // for (int c = 0; c < speciesCount; c++) ...
100        return result;
101    } // public double evaluate(double x, IRealVector systemValues) ...
102
103    /**
104     * Wertet die partiell nach der Spezies mit dem angegebenen
105     * Index abgeleitete erzeugende Funktion an der angegebenen
106     * Stelle aus.
107     * @param index Index der Spezies, nach der abgeleitet werden soll.
108     * @param x Wert, an dem die erzeugende Funktion ausgewertet werden soll.
109     * @param systemValues Vektor der Werte der exponentiellen
110     *   erzeugenden Funktionen des Systems. (Index entspricht dem
111     *   Index der Spezies in der Spezifikation)
112     * @return Wert der partiellen Ableitung an der angegebenen Stelle.
113     * @throws IllegalArgumentException Wird geworfen,
114     *   wenn der angegebene Vektor null ist, nicht die richtige
115     *   Dimension besitzt, oder der Parameter
116     *   ausserhalb des Definitionsbereichs der erzeugenden
117     *   Funktion liegt.
118     * @throws IndexOutOfBoundsException wird geworfen,
119     *   wenn der angegebene Index ausserhalb des gueltigen Bereichs liegt.
120     */
121    @Override
122    public double evaluatePartialDerivative(int index, double x, IRealVector systemValues) {
123        if (index < 0 || index >= _speciesCount) {
124            throw new IndexOutOfBoundsException(EXCEPTION_MESSAGE_SPECIES_INDEX);
125        } // if (index < 0 || index >= _speciesCount) ...
126        if (systemValues == null || systemValues.getDimension() != _speciesCount) {
127            throw new IllegalArgumentException(EXCEPTION_MESSAGE_SYSTEM_VECTOR_INVALID);
128        } // if (systemValues == null || systemValues.getDimension() != _speciesCount) ...
129        int[] speciesIndexCount = _speciesIndexCount;
130        if (speciesIndexCount[index] > 0) {
131            double result = 1.0D;
132            // Singleton-Spezies behandeln
133            int currentExponent = _speciesSingletonCount;
134            if (currentExponent > 0) {
135                result *= Math.pow(x, currentExponent);
136            } // if (currentExponent > 0) ...
137            int speciesCount = _speciesCount;
138            for (int c = 0; c < speciesCount; c++) {
139                currentExponent = speciesIndexCount[c];
140                if (currentExponent > 0) {
141                    if (c == index) {
142                        if (currentExponent > 1) {
143                          result *= (double) currentExponent * Math.pow(systemValues.getEntry(c
                                ), currentExponent - 1);
144                        } // if (currentExponent > 1) ...
145                    } else { // if (c != index) ...
146                        result *= Math.pow(systemValues.getEntry(c), currentExponent);
147                    } // if (c != index) ...
148                } // if (currentExponent > 0) ...
149            } // for (int c = 0; c < speciesCount; c++) ...
150            return result;
151        } else { // if (speciesIndexCount[index] <= 0) ...
152            return 0.0D;
153        } // if (speciesIndexCount[index] <= 0) ...
154    } // public double evaluatePartialDerivative(int index, double x, IRealVector systemValues)
          ...
155
156    /**
157     * Gibt eine String-Repraesentation der Instanz zurueck.
```

```
158      * @return String-Repraesentation der Instanz.
159      */
160     @Override
161     public String toString() {
162         StringBuilder sb = new StringBuilder();
163         sb.append("  Produkt ");
164         boolean output = false;
165         if (_speciesSingletonCount > 0) {
166             if (output) {
167                 sb.append(" * ");
168             }
169             sb.append("Z^").append(_speciesSingletonCount);
170             output = true;
171         } // if (_speciesSingletonCount > 0) ...
172         for (int c = 0; c < _speciesCount; c++) {
173             int currentCount = _speciesIndexCount[c];
174             if (currentCount > 0) {
175                 if (output) {
176                     sb.append(" * ");
177                 }
178                 sb.append("Y_").append(c).append("^").append(currentCount);
179                 output = true;
180             } // if (currentCount > 0) ...
181         } // for (int c = 0; c < speciesCount; c++) ...
182         sb.append("\n");
183         return sb.toString();
184     } // public String toString() ...
185 } // class ExponentialGeneratingFunctionProduct extends ExponentialGeneratingFunction ...
```

Listing 34: Klasse *at.techmath.boltzmann.oracle.ExponentialGeneratingFunctionSequence*

```
1  package at.techmath.boltzmann.oracle;
2
3  import at.techmath.boltzmann.linearalgebra.IRealVector;
4
5  /**
6   * Exponentielle erzeugende Funktion der Sequence-Species,
7   * in die eine benannte Spezies eingesetzt wird.
8   * @author Stefan Schnabl (e0226245)
9   */
10 class ExponentialGeneratingFunctionSequence extends ExponentialGeneratingFunction {
11     /**
12      * Meldung der Exception, die anzeigt,
13      * dass der Parameter ausserhalb des Konvergenzradiuses liegt.
14      */
15     protected static final String EXCEPTION_MESSAGE_RADIUS_OF_CONVERGENCE = "Parameter
           ausserhalb des Konvergenzradius der Sequence-EEF.";
16
17     /**
18      * Erzeugt eine neue Instanz.
19      * @param speciesIndex Index der Spezies.
20      * @param speciesCount Anzahl der Spezies.
21      */
22     public ExponentialGeneratingFunctionSequence(int speciesIndex, int speciesCount) {
23         super();
24         _speciesIndex = speciesIndex;
25         _speciesCount = speciesCount;
26     } // public ExponentialGeneratingFunctionSequence(int speciesIndex, int speciesCount) ...
27
28     /**
29      * Index der Spezies, die eingesetzt wird.
30      */
31     private int _speciesIndex;
32
33     /**
34      * Anzahl der Spezies.
35      */
36     private int _speciesCount;
37
38     /**
39      * Wertet die exponentielle erzeugende Funktion an der
40      * uebergebenen Stelle aus.
41      * @param x Stelle, an der die EEF ausgewertet werden soll.
42      * @param systemValues Vektor der Werte der exponentiellen
43      *   erzeugenden Funktionen des Systems. (Index entspricht dem
44      *   Index der Spezies in der Spezifikation)
45      * @return Wert der exponentiellen erzeugenden Funktion an
46      *   der angegebenen Stelle.
47      * @throws IllegalArgumentException Wird geworfen,
48      *   wenn der angegebene Vektor null ist, nicht die richtige
49      *   Dimension besitzt, oder der Parameter
50      *   ausserhalb des Definitionsbereichs der erzeugenden
51      *   Funktion liegt.
52      */
53     @Override
54     public double evaluate(double x, IRealVector systemValues) {
55         if (systemValues == null || systemValues.getDimension() != _speciesCount) {
56             throw new IllegalArgumentException(EXCEPTION_MESSAGE_SYSTEM_VECTOR_INVALID);
57         } // if (systemValues == null || systemValues.getDimension() <= _speciesIndex) ...
58         double value = systemValues.getEntry(_speciesIndex);
59         if (value < 0.0D || value >= 1.0D) {
60             throw new IllegalArgumentException(EXCEPTION_MESSAGE_RADIUS_OF_CONVERGENCE);
61         } // if (value < 0.0D || value >= 1.0D) ...
62         return 1.0D / (1.0D - value);
63     } // public double evaluate(double x, IRealVector systemValues) ...
64
65     /**
66      * Wertet die partiell nach der Spezies mit dem angegebenen
67      * Index abgeleitete erzeugende Funktion an der angegebenen
68      * Stelle aus.
69      * @param index Index der Spezies, nach der abgeleitet werden soll.
70      * @param x Wert, an dem die erzeugende Funktion ausgewertet werden soll.
71      * @param systemValues Vektor der Werte der exponentiellen
72      *   erzeugenden Funktionen des Systems. (Index entspricht dem
73      *   Index der Spezies in der Spezifikation)
74      * @return Wert der partiellen Ableitung an der angegebenen Stelle.
75      * @throws IllegalArgumentException Wird geworfen,
76      *   wenn der angegebene Vektor null ist, nicht die richtige
77      *   Dimension besitzt, oder der Parameter
78      *   ausserhalb des Definitionsbereichs der erzeugenden
```

```
79      *   Funktion liegt.
80      * @throws IndexOutOfBoundsException wird geworfen,
81      *   wenn der angegebene Index ausserhalb des gueltigen Bereichs liegt.
82      */
83     @Override
84     public double evaluatePartialDerivative(int index, double x, IRealVector systemValues) {
85         if (index < 0 || index >= _speciesCount) {
86             throw new IndexOutOfBoundsException(EXCEPTION_MESSAGE_SPECIES_INDEX);
87         } // if (index < 0 || index >= _speciesCount) ...
88         if (systemValues == null || systemValues.getDimension() != _speciesCount) {
89             throw new IllegalArgumentException(EXCEPTION_MESSAGE_SYSTEM_VECTOR_INVALID);
90         } // if (systemValues == null || systemValues.getDimension() != _speciesCount) ...
91         if (index == _speciesIndex) {
92             double value = systemValues.getEntry(_speciesIndex);
93             if (value < 0.0D || value >= 1.0D) {
94                 throw new IllegalArgumentException(EXCEPTION_MESSAGE_RADIUS_OF_CONVERGENCE);
95             } // if (value < 0.0D || value >= 1.0D) ...
96             double numeratorRoot = 1.0D - value;
97             return 1.0D / (numeratorRoot * numeratorRoot);
98         } else { // if (index != _speciesIndex) ...
99             return 0.0D;
100        } // if (index != _speciesIndex) ...
101    } // public double evaluatePartialDerivative(int index, double x, IRealVector systemValues)
          ...
102
103    /**
104     * Gibt eine String-Repraesentation der Instanz zurueck.
105     * @return String-Repraesentation der Instanz.
106     */
107    @Override
108    public String toString() {
109        StringBuilder sb = new StringBuilder();
110        sb.append("  Sequence ");
111        sb.append("Y_").append(_speciesIndex);
112        sb.append("\n");
113        return sb.toString();
114    } // public String toString() ...
115 } // class ExponentialGeneratingFunctionSequence extends ExponentialGeneratingFunction ...
```

Listing 35: Klasse *at.techmath.boltzmann.oracle.ExponentialGeneratingFunctionSequenceSingleton*

```
1  package at.techmath.boltzmann.oracle;
2
3  import at.techmath.boltzmann.linearalgebra.IRealVector;
4
5  /**
6   * Exponentielle erzeugende Funktion der Sequence-Species,
7   * in die die Singleton-Spezies eingesetzt wird.
8   * @author Stefan Schnabl (e0226245)
9   */
10 class ExponentialGeneratingFunctionSequenceSingleton extends ExponentialGeneratingFunction {
11     /**
12      * Meldung der Exception, die anzeigt,
13      * dass der Parameter ausserhalb des Konvergenzradiuses liegt.
14      */
15     protected static final String EXCEPTION_MESSAGE_RADIUS_OF_CONVERGENCE = "Parameter
           ausserhalb des Konvergenzradius der Sequence-EEF.";
16
17     /**
18      * Erzeugt eine neue Instanz.
19      * @param speciesCount Anzahl der Spezies.
20      */
21     public ExponentialGeneratingFunctionSequenceSingleton(int speciesCount) {
22         super();
23         _speciesCount = speciesCount;
24     } // public ExponentialGeneratingFunctionSequenceSingleton(int speciesCount) ...
25
26     /**
27      * Anzahl der Spezies.
28      */
29     private int _speciesCount;
30
31     /**
32      * Wertet die exponentielle erzeugende Funktion an der
33      * uebergebenen Stelle aus.
34      * @param x Stelle, an der die EEF ausgewertet werden soll.
35      * @param systemValues Vektor der Werte der exponentiellen
36      *   erzeugenden Funktionen des Systems. (Index entspricht dem
37      *   Index der Spezies in der Spezifikation)
38      * @return Wert der exponentiellen erzeugenden Funktion an
39      *   der angegebenen Stelle.
40      * @throws IllegalArgumentException Wird geworfen,
41      *   wenn der angegebene Vektor null ist, nicht die richtige
42      *   Dimension besitzt, oder der Parameter
43      *   ausserhalb des Definitionsbereichs der erzeugenden
44      *   Funktion liegt.
45      */
46     @Override
47     public double evaluate(double x, IRealVector systemValues) {
48         if (systemValues == null || systemValues.getDimension() != _speciesCount) {
49             throw new IllegalArgumentException(EXCEPTION_MESSAGE_SYSTEM_VECTOR_INVALID);
50         } // if (systemValues == null || systemValues.getDimension() <= _speciesIndex) ...
51         if (x < 0.0D || x >= 1.0D) {
52             throw new IllegalArgumentException(EXCEPTION_MESSAGE_RADIUS_OF_CONVERGENCE);
53         } // if (x < 0.0D || x >= 1.0D) ...
54         return 1.0D / (1.0D - x);
55     } // public double evaluate(double x, IRealVector systemValues) ...
56
57     /**
58      * Wertet die partiell nach der Spezies mit dem angegebenen
59      * Index abgeleitete erzeugende Funktion an der angegebenen
60      * Stelle aus.
61      * @param index Index der Spezies, nach der abgeleitet werden soll.
62      * @param x Wert, an dem die erzeugende Funktion ausgewertet werden soll.
63      * @param systemValues Vektor der Werte der exponentiellen
64      *   erzeugenden Funktionen des Systems. (Index entspricht dem
65      *   Index der Spezies in der Spezifikation)
66      * @return Wert der partiellen Ableitung an der angegebenen Stelle.
67      * @throws IllegalArgumentException Wird geworfen,
68      *   wenn der angegebene Vektor null ist, nicht die richtige
69      *   Dimension besitzt, oder der Parameter
70      *   ausserhalb des Definitionsbereichs der erzeugenden
71      *   Funktion liegt.
72      * @throws IndexOutOfBoundsException wird geworfen,
73      *   wenn der angegebene Index ausserhalb des gueltigen Bereichs liegt.
74      */
75     @Override
76     public double evaluatePartialDerivative(int index, double x, IRealVector systemValues) {
77         if (index < 0 || index >= _speciesCount) {
78             throw new IndexOutOfBoundsException(EXCEPTION_MESSAGE_SPECIES_INDEX);
```

```
79          } // if (index < 0 || index >= _speciesCount) ...
80          if (systemValues == null || systemValues.getDimension() != _speciesCount) {
81              throw new IllegalArgumentException(EXCEPTION_MESSAGE_SYSTEM_VECTOR_INVALID);
82          } // if (systemValues == null || systemValues.getDimension() != _speciesCount) ...
83          return 0.0D;
84      } // public double evaluatePartialDerivative(int index, double x, IRealVector systemValues)
            ...
85
86      /**
87       * Gibt eine String-Repraesentation der Instanz zurueck.
88       * @return String-Repraesentation der Instanz.
89       */
90      @Override
91      public String toString() {
92          StringBuilder sb = new StringBuilder();
93          sb.append("  Sequence ");
94          sb.append("Z");
95          sb.append("\n");
96          return sb.toString();
97      } // public String toString() ...
98  } // class ExponentialGeneratingFunctionSequenceSingleton extends ExponentialGeneratingFunction
        ...
```

Listing 36: Klasse *at.techmath.boltzmann.oracle.ExponentialGeneratingFunctionSet*

```
1  package at.techmath.boltzmann.oracle;
2
3  import at.techmath.boltzmann.linearalgebra.IRealVector;
4
5  /**
6   * Exponentielle erzeugende Funktion der Set-Species,
7   * in die eine benannte Spezies eingesetzt wird.
8   * @author Stefan Schnabl (e0226245)
9   */
10 class ExponentialGeneratingFunctionSet extends ExponentialGeneratingFunction {
11     /**
12      * Erzeugt eine neue Instanz.
13      * @param speciesIndex Index der Spezies.
14      * @param speciesCount Anzahl der Spezies.
15      */
16     public ExponentialGeneratingFunctionSet(int speciesIndex, int speciesCount) {
17         super();
18         _speciesIndex = speciesIndex;
19         _speciesCount = speciesCount;
20     } // public ExponentialGeneratingFunctionSet(int speciesIndex, int speciesCount) ...
21
22     /**
23      * Index der Spezies, die eingesetzt wird.
24      */
25     private int _speciesIndex;
26
27     /**
28      * Anzahl der Spezies.
29      */
30     private int _speciesCount;
31
32     /**
33      * Wertet die exponentielle erzeugende Funktion an der
34      * uebergebenen Stelle aus.
35      * @param x Stelle, an der die EEF ausgewertet werden soll.
36      * @param systemValues Vektor der Werte der exponentiellen
37      *   erzeugenden Funktionen des Systems. (Index entspricht dem
38      *   Index der Spezies in der Spezifikation)
39      * @return Wert der exponentiellen erzeugenden Funktion an
40      *   der angegebenen Stelle.
41      * @throws IllegalArgumentException Wird geworfen,
42      *   wenn der angegebene Vektor null ist, nicht die richtige
43      *   Dimension besitzt, oder der Parameter
44      *   ausserhalb des Definitionsbereichs der erzeugenden
45      *   Funktion liegt.
46      */
47     @Override
48     public double evaluate(double x, IRealVector systemValues) {
49         if (systemValues == null || systemValues.getDimension() != _speciesCount) {
50             throw new IllegalArgumentException(EXCEPTION_MESSAGE_SYSTEM_VECTOR_INVALID);
51         } // if (systemValues == null || systemValues.getDimension() != _speciesCount) ...
52         return Math.exp(systemValues.getEntry(_speciesIndex));
53     } // public double evaluate(double x, IRealVector systemValues) ...
54
55     /**
56      * Wertet die partiell nach der Spezies mit dem angegebenen
57      * Index abgeleitete erzeugende Funktion an der angegebenen
58      * Stelle aus.
59      * @param index Index der Spezies, nach der abgeleitet werden soll.
60      * @param x Wert, an dem die erzeugende Funktion ausgewertet werden soll.
61      * @param systemValues Vektor der Werte der exponentiellen
62      *   erzeugenden Funktionen des Systems. (Index entspricht dem
63      *   Index der Spezies in der Spezifikation)
64      * @return Wert der partiellen Ableitung an der angegebenen Stelle.
65      * @throws IllegalArgumentException Wird geworfen,
66      *   wenn der angegebene Vektor null ist, nicht die richtige
67      *   Dimension besitzt, oder der Parameter
68      *   ausserhalb des Definitionsbereichs der erzeugenden
69      *   Funktion liegt.
70      * @throws IndexOutOfBoundsException wird geworfen,
71      *   wenn der angegebene Index ausserhalb des gueltigen Bereichs liegt.
72      */
73     @Override
74     public double evaluatePartialDerivative(int index, double x, IRealVector systemValues) {
75         if (index < 0 || index >= _speciesCount) {
76             throw new IndexOutOfBoundsException(EXCEPTION_MESSAGE_SPECIES_INDEX);
77         } // if (index < 0 || index >= _speciesCount) ...
78         if (systemValues == null || systemValues.getDimension() != _speciesCount) {
79             throw new IllegalArgumentException(EXCEPTION_MESSAGE_SYSTEM_VECTOR_INVALID);
```

```
80          } // if (systemValues == null || systemValues.getDimension() != _speciesCount) ...
81          if (index == _speciesIndex) {
82              return Math.exp(systemValues.getEntry(_speciesIndex));
83          } else { // if (index != _speciesIndex) ...
84              return 0.0D;
85          } // if (index != _speciesIndex) ...
86      } // public double evaluatePartialDerivative(int index, double x, IRealVector systemValues)
            ...
87
88      /**
89       * Gibt eine String-Repraesentation der Instanz zurueck.
90       * @return String-Repraesentation der Instanz.
91       */
92      @Override
93      public String toString() {
94          StringBuilder sb = new StringBuilder();
95          sb.append("  Set ");
96          sb.append("Y_").append(_speciesIndex);
97          sb.append("\n");
98          return sb.toString();
99      } // public String toString() ...
100 } // class ExponentialGeneratingFunctionSet extends ExponentialGeneratingFunction ...
```

Listing 37: Klasse *at.techmath.boltzmann.oracle.ExponentialGeneratingFunctionSetSingleton*

```
1  package at.techmath.boltzmann.oracle;
2
3  import at.techmath.boltzmann.linearalgebra.IRealVector;
4
5  /**
6   * Exponentielle erzeugende Funktion der Set-Species,
7   * in die die Singleton-Spezies eingesetzt wird.
8   * @author Stefan Schnabl (e0226245)
9   */
10 class ExponentialGeneratingFunctionSetSingleton extends ExponentialGeneratingFunction {
11     /**
12      * Erzeugt eine neue Instanz.
13      * @param speciesCount Anzahl der Spezies.
14      */
15     public ExponentialGeneratingFunctionSetSingleton(int speciesCount) {
16         super();
17         _speciesCount = speciesCount;
18     } // public ExponentialGeneratingFunctionSetSingleton(int speciesCount) ...
19
20     /**
21      * Anzahl der Spezies.
22      */
23     private int _speciesCount;
24
25     /**
26      * Wertet die exponentielle erzeugende Funktion an der
27      * uebergebenen Stelle aus.
28      * @param x Stelle, an der die EEF ausgewertet werden soll.
29      * @param systemValues Vektor der Werte der exponentiellen
30      *   erzeugenden Funktionen des Systems. (Index entspricht dem
31      *   Index der Spezies in der Spezifikation)
32      * @return Wert der exponentiellen erzeugenden Funktion an
33      *   der angegebenen Stelle.
34      * @throws IllegalArgumentException Wird geworfen,
35      *   wenn der angegebene Vektor null ist, nicht die richtige
36      *   Dimension besitzt, oder der Parameter
37      *   ausserhalb des Definitionsbereichs der erzeugenden
38      *   Funktion liegt.
39      */
40     @Override
41     public double evaluate(double x, IRealVector systemValues) {
42         if (systemValues == null || systemValues.getDimension() != _speciesCount) {
43             throw new IllegalArgumentException(EXCEPTION_MESSAGE_SYSTEM_VECTOR_INVALID);
44         } // if (systemValues == null || systemValues.getDimension() != _speciesCount) ...
45         return Math.exp(x);
46     } // public double evaluate(double x, IRealVector systemValues) ...
47
48     /**
49      * Wertet die partiell nach der Spezies mit dem angegebenen
50      * Index abgeleitete erzeugende Funktion an der angegebenen
51      * Stelle aus.
52      * @param index Index der Spezies, nach der abgeleitet werden soll.
53      * @param x Wert, an dem die erzeugende Funktion ausgewertet werden soll.
54      * @param systemValues Vektor der Werte der exponentiellen
55      *   erzeugenden Funktionen des Systems. (Index entspricht dem
56      *   Index der Spezies in der Spezifikation)
57      * @return Wert der partiellen Ableitung an der angegebenen Stelle.
58      * @throws System.IllegalArgumentException Wird geworfen,
59      *   wenn der angegebene Vektor null ist, nicht die richtige
60      *   Dimension besitzt, oder der Parameter
61      *   ausserhalb des Definitionsbereichs der erzeugenden
62      *   Funktion liegt.
63      * @throws IndexOutOfBoundsException wird geworfen,
64      *   wenn der angegebene Index ausserhalb des gueltigen Bereichs liegt.
65      */
66     @Override
67     public double evaluatePartialDerivative(int index, double x, IRealVector systemValues) {
68         if (index < 0 || index >= _speciesCount) {
69             throw new IndexOutOfBoundsException(EXCEPTION_MESSAGE_SPECIES_INDEX);
70         } // if (index < 0 || index >= _speciesCount) ...
71         if (systemValues == null || systemValues.getDimension() != _speciesCount) {
72             throw new IllegalArgumentException(EXCEPTION_MESSAGE_SYSTEM_VECTOR_INVALID);
73         } // if (systemValues == null || systemValues.getDimension() != _speciesCount) ...
74         return 0.0D;
75     } // public double evaluatePartialDerivative(int index, double x, IRealVector systemValues)
          ...
76
77     /**
78      * Gibt eine String-Repraesentation der Instanz zurueck.
```

```
79      * @return String-Repraesentation der Instanz.
80      */
81     @Override
82     public String toString() {
83         StringBuilder sb = new StringBuilder();
84         sb.append("  Set ");
85         sb.append("Z");
86         sb.append("\n");
87         return sb.toString();
88     } // public String toString() ...
89 } // class ExponentialGeneratingFunctionSetSingleton extends ExponentialGeneratingFunction ...
```

Listing 38: Klasse *at.techmath.boltzmann.oracle.ExponentialGeneratingFunctionSum*

```
1  package at.techmath.boltzmann.oracle;
2
3  import at.techmath.boltzmann.linearalgebra.IRealVector;
4
5  /**
6   * Exponentielle erzeugende Funktion der Summen-Spezies.
7   * @author Stefan Schnabl (e0226245)
8   */
9  class ExponentialGeneratingFunctionSum extends ExponentialGeneratingFunction {
10     /**
11      * Erzeugt eine neue Instanz.
12      * @param speciesCount Anzahl der Spezies.
13      * @throws IllegalArgumentException Wird geworfen, wenn der speciesCount
14      *   negativ ist.
15      */
16     public ExponentialGeneratingFunctionSum(int speciesCount) {
17         super();
18         if (speciesCount < 0) {
19             throw new IllegalArgumentException("speciesCount < 0.");
20         } // if (speciesCount < 0) ...
21         this._speciesCount = speciesCount;
22         this._speciesOneCount = 0;
23         this._speciesSingletonCount = 0;
24         int[] speciesIndexCount = this._speciesIndexCount = new int[speciesCount];
25         for (int c = 0; c < speciesCount; c++) {
26             speciesIndexCount[c] = 0;
27         } // for (int c = 0; c < speciesCount; c++) ...
28     } // public ExponentialGeneratingFunctionSum(int speciesCount) ...
29
30     /**
31      * Anzahl der Spezies.
32      */
33     private int _speciesCount;
34
35     /**
36      * Anzahl der Summanden, die die One-Spezies sind.
37      */
38     private int _speciesOneCount;
39
40     /**
41      * Erhoeht die Anzahl der Summanden, die die One-Spezies sind, um eins.
42      */
43     protected void incrementSpeciesOneCount() {
44         _speciesOneCount += 1;
45     } // protected void incrementSpeciesOneCount() ...
46
47     /**
48      * Anzahl der Summanden, die die Singleton-Spezies sind.
49      */
50     private int _speciesSingletonCount;
51
52     /**
53      * Erhoeht die Anzahl der Summanden, die die Singleton-Spezies sind, um eins.
54      */
55     protected void incrementSpeciesSingletonCount() {
56         _speciesSingletonCount += 1;
57     } // protected void incrementSpeciesSingletonCount() ...
58
59     /**
60      * Anzahl der Summanden, die benannte Spezies sind.
61      * Der Index ins Array entspricht dem Index der benannten Spezies.
62      */
63     private int[] _speciesIndexCount;
64
65     /**
66      * Erhoeht die Anzahl der Summanden, die die benannte Spezies mit
67      * dem angegebenen Index sind, um eins.
68      * @param index Index der Spezies.
69      * @throws IndexOutOfBoundsException Wird geworfen, wenn der Index
70      *   ausserhalb des gueltigen Bereichs liegt.
71      */
72     protected void incrementSpeciesIndexCount(int index) {
73         if (index < 0 || index >= _speciesCount) {
74             throw new IndexOutOfBoundsException("Index ausserhalb des gueltigen Bereichs.");
75         } // if (index < 0 || index >= _speciesCount) ...
76         _speciesIndexCount[index] += 1;
77     } // protected void incrementSpeciesIndexCount(int index) ...
78
79     /**
```

```
80       * Wertet die exponentielle erzeugende Funktion an der
81       * uebergebenen Stelle aus.
82       * @param x Stelle, an der die EEF ausgewertet werden soll.
83       * @param systemValues Vektor der Werte der exponentiellen
84       *   erzeugenden Funktionen des Systems. (Index entspricht dem
85       *   Index der Spezies in der Spezifikation)
86       * @return Wert der exponentiellen erzeugenden Funktion an
87       *   der angegebenen Stelle.
88       * @throws IllegalArgumentException Wird geworfen,
89       *   wenn der angegebene Vektor null ist, nicht die richtige
90       *   Dimension besitzt, oder der Parameter
91       *   ausserhalb des Definitionsbereichs der erzeugenden
92       *   Funktion liegt.
93       */
94      @Override
95      public double evaluate(double x, IRealVector systemValues) {
96          if (systemValues == null || systemValues.getDimension() != _speciesCount) {
97              throw new IllegalArgumentException(EXCEPTION_MESSAGE_SYSTEM_VECTOR_INVALID);
98          } // if (systemValues == null || systemValues.getDimension() != _speciesCount) ...
99          double result = 0.0D;
100         // One-Spezies behandeln
101         int currentCoefficient = _speciesOneCount;
102         if (currentCoefficient > 0) {
103             result += (double) currentCoefficient;
104         } // if (currentCoefficient > 0) ...
105         // Singleton-Spezies behandeln
106         currentCoefficient = _speciesSingletonCount;
107         if (currentCoefficient > 0) {
108             result += (double) currentCoefficient * x;
109         } // if (currentCoefficient > 0) ...
110         int speciesCount = _speciesCount;
111         int[] speciesIndexCount = _speciesIndexCount;
112         for (int c = 0; c < speciesCount; c++) {
113             currentCoefficient = speciesIndexCount[c];
114             if (currentCoefficient > 0) {
115                 result += (double) currentCoefficient * systemValues.getEntry(c);
116             } // if (currentCoefficient > 0) ...
117         } // for (int c = 0; c < speciesCount; c++) ...
118         return result;
119     } // public double evaluate(double x, IRealVector systemValues) ...
120
121     /**
122      * Wertet die partiell nach der Spezies mit dem angegebenen
123      * Index abgeleitete erzeugende Funktion an der angegebenen
124      * Stelle aus.
125      * @param index Index der Spezies, nach der abgeleitet werden soll.
126      * @param x Wert, an dem die erzeugende Funktion ausgewertet werden soll.
127      * @param systemValues Vektor der Werte der exponentiellen
128      *   erzeugenden Funktionen des Systems. (Index entspricht dem
129      *   Index der Spezies in der Spezifikation)
130      * @return Wert der partiellen Ableitung an der angegebenen Stelle.
131      * @throws IllegalArgumentException Wird geworfen,
132      *   wenn der angegebene Vektor null ist, nicht die richtige
133      *   Dimension besitzt, oder der Parameter
134      *   ausserhalb des Definitionsbereichs der erzeugenden
135      *   Funktion liegt.
136      * @throws IndexOutOfBoundsException wird geworfen,
137      *   wenn der angegebene Index ausserhalb des gueltigen Bereichs liegt.
138      */
139     @Override
140     public double evaluatePartialDerivative(int index, double x, IRealVector systemValues) {
141         if (index < 0 || index >= _speciesCount) {
142             throw new IndexOutOfBoundsException(EXCEPTION_MESSAGE_SPECIES_INDEX);
143         } // if (index < 0 || index >= _speciesCount) ...
144         if (systemValues == null || systemValues.getDimension() != _speciesCount) {
145             throw new IllegalArgumentException(EXCEPTION_MESSAGE_SYSTEM_VECTOR_INVALID);
146         } // if (systemValues == null || systemValues.getDimension() != _speciesCount) ...
147         // Die Ableitung der Summe i = 1 bis m von Lambda_i Y_i nach Y_i = Lambda_i.
148         return (double) _speciesIndexCount[index];
149     } // public double evaluatePartialDerivative(int index, double x, IRealVector systemValues)
          ...
150
151     /**
152      * Gibt eine String-Repraesentation der Instanz zurueck.
153      * @return String-Repraesentation der Instanz.
154      */
155     @Override
156     public String toString() {
157         StringBuilder sb = new StringBuilder();
158         sb.append("  Summe ");
```

```
159         boolean output = false;
160         if (_speciesOneCount > 0) {
161             sb.append(_speciesOneCount).append("*1");
162             output = true;
163         } // if (_speciesOneCount > 0) ...
164         if (_speciesSingletonCount > 0) {
165             if (output) {
166                 sb.append(" + ");
167             }
168             sb.append(_speciesSingletonCount).append("*Z");
169             output = true;
170         } // if (_speciesSingletonCount > 0) ...
171         for (int c = 0; c < _speciesCount; c++) {
172             int currentCount = _speciesIndexCount[c];
173             if (currentCount > 0) {
174                 if (output) {
175                     sb.append(" + ");
176                 }
177                 sb.append(currentCount).append("*Y_").append(c);
178                 output = true;
179             } // if (currentCount > 0) ...
180         } // for (int c = 0; c < speciesCount; c++) ...
181         sb.append("\n");
182         return sb.toString();
183     } // public String toString() ...
184 } // class EpxonentialGeneratingFunctionSum extends ExponentialGeneratingFunction ...
```

Listing 39: Interface *at.techmath.boltzmann.oracle.IExponentialBoltzmannOracle*

```
1  package at.techmath.boltzmann.oracle;
2
3  import at.techmath.boltzmann.specification.ISpecification;
4
5  /**
6   * Repraesentiert ein Boltzmann-Orakel zur
7   * Auswertung der exponentiellen erzeugenden
8   * Funktionen einer kombinatorischen Spezifiaktion.
9   * @author Stefan Schnabl (e0226245)
10  */
11 public interface IExponentialBoltzmannOracle {
12     /**
13      * Wertet die erzeugende Funktion der Spezifikation
14      * des Orakels fuer die Spezies mit dem angegebenen
15      * Index fuer den angegebenen Parameter aus.
16      * @param index Index der Spezies innerhalb der Spezifikation.
17      * @param x Parameter der exponentiellen erzeugenden Funktion.
18      * @return Wert der erzeugenden Funktion der angegebenen
19      *   Spezies an der angegebenen Stelle.
20      */
21     double evaluate(int index, double x);
22
23     /**
24      * Gibt die Spezifikation des Orakels zurueck.
25      * @return Spezifikation des Orakels.
26      */
27     ISpecification getSpecification();
28 } // public interface IExponentialBoltzmannOracle ...
```

Listing 40: Interface *at.techmath.boltzmann.oracle.IExponentialBoltzmannOracleFactory*

```
1  package at.techmath.boltzmann.oracle;
2
3  import at.techmath.boltzmann.specification.ISpecification;
4
5  /**
6   * Factory, die aus kombinatorischen Spezifikationen
7   * exponentielle Boltzmann-Orakel erzeugen kann.
8   * @author Stefan Schnabl (e0226245)
9   */
10 public interface IExponentialBoltzmannOracleFactory {
11     /**
12      * Erzeugt fuer die angegebene Spezifikation
13      * ein exponentielles Boltzmann-Orakel.
14      * @param specification Spezifikation, fuer die das Orakel erzeugt werden soll.
15      * @return Exponentielles Boltzmann-Orakel fuer die angegebene Spezifikation.
16      * @throws IllegalArgumentException Wird geworfen, wenn die
17      *   uebergebene Spezifikation null ist.
18      */
19     IExponentialBoltzmannOracle createOracle(ISpecification specification);
20 } // public interface IExponentialBoltzmannOracleFactor ...
```

A.1.4 Package at.techmath.boltzmann.output

Listing 41: Interface *at.techmath.boltzmann.output.IStructureOutput*

```
1  package at.techmath.boltzmann.output;
2
3  import java.io.OutputStream;
4  import java.io.IOException;
5  import at.techmath.boltzmann.sampler.ISamplingResult;
6
7  /**
8   * Dieses Interface definiert Methoden zur
9   * Ausgabe von gesampelten Strukturen.
10  * @author Stefan Schnabl (e0226245)
11  */
12 public interface IStructureOutput {
13      /**
14       * Gibt die angegebene gesampelte Struktur in den angegebenen Stream aus.
15       * @param samplingResult Gesampelte Struktur, die ausgegeben werden soll.
16       * @param outputStream Stream, in den ausgegeben werden soll.
17       * @throws java.io.IOException Fehler bei der Stream-Ausgabe
18       * @throws IllegalArgumentException samplingResult oder outputStream ist null.
19       */
20      void outputSamplingResult(ISamplingResult samplingResult, OutputStream outputStream) throws
            IOException;
21 } // public interface IStructureOutput ...
```

Listing 42: Klasse *at.techmath.boltzmann.output.StructureOutputText*

```
1  package at.techmath.boltzmann.output;
2
3  import java.io.IOException;
4  import java.io.OutputStream;
5  import java.io.PrintStream;
6
7  import at.techmath.boltzmann.sampler.ISamplingResult;
8
9  /**
10  * Gibt Strukturen als Text aus.
11  * @author Stefan Schnabl (e0226245)
12  */
13 public class StructureOutputText implements IStructureOutput {
14     /**
15      * Gibt die angegebene gesampelte Struktur in den angegebenen Stream aus.
16      * @param samplingResult Gesampelte Struktur, die ausgegeben werden soll.
17      * @param outputStream Stream, in den ausgegeben werden soll.
18      * @throws java.io.IOException Fehler bei der Stream-Ausgabe
19      * @throws IllegalArgumentException samplingResult oder outputStream ist null.
20      */
21     public void outputSamplingResult(ISamplingResult samplingResult, OutputStream outputStream)
            throws IOException {
22         if (samplingResult == null) {
23             throw new IllegalArgumentException("samplingResult = null.");
24         } // if (samplingResult == null) ...
25         if (outputStream == null) {
26             throw new IllegalArgumentException("outputStream = null.");
27         } // if (outputStream == null) ...
28         PrintStream printStream = null;
29         try {
30             printStream = new PrintStream(outputStream, false);
31             StructureOutputTextContext context = new StructureOutputTextContext(printStream);
32             StructureOutputTextMethod initialMethod = context.getOutputTextMethod(
                    samplingResult.getStructure());
33             context.pushMethod(initialMethod);
34             while (!context.isEmpty()) {
35                 StructureOutputTextMethod currentMethod = context.peekMethod();
36                 currentMethod.execute(context);
37             } // while (!context.isEmpty()) ..
38             printStream.println(";");
39         } finally {
40             if (printStream != null) {
41                 printStream.flush();
42             } // if (printStream != null) ...
43         }
44     } // public void outputSamplingResult(ISamplingResult samplingResult, OutputStream
           outputStream) throws IOException ...
45 } // public class StructureOutputText implements IStructureOutput ...
```

Listing 43: Klasse *at.techmath.boltzmann.output.StructureOutputTextContext*

```
1  package at.techmath.boltzmann.output;
2
3  import java.io.PrintStream;
4  import java.util.Stack;
5
6  import at.techmath.boltzmann.sampler.IStructure;
7  import at.techmath.boltzmann.sampler.IStructureIntegerSingleton;
8  import at.techmath.boltzmann.sampler.IStructureSingleton;
9  import at.techmath.boltzmann.sampler.IStructureOne;
10 import at.techmath.boltzmann.sampler.IStructureSum;
11 import at.techmath.boltzmann.sampler.IStructureProduct;
12 import at.techmath.boltzmann.sampler.IStructureSequence;
13 import at.techmath.boltzmann.sampler.IStructureSet;
14 import at.techmath.boltzmann.sampler.IStructureCycle;
15
16 /**
17  * Context-Klasse zur Ausgabe einer Struktur als Text.
18  * @author Stefan Schnabl (e0226245)
19  */
20 class StructureOutputTextContext {
21     /**
22      * Erzeugt eine neue Instanz.
23      * @param printStream PrintStream-Instanz, in die die Ausgabe erfolgen soll.
24      * @throws IllegalArgumentException printStream ist null.
25      */
26     public StructureOutputTextContext(PrintStream printStream) {
27         super();
28         if (printStream == null) {
29             throw new IllegalArgumentException("printStream = null.");
30         } // if (printStream == null) ...
31         _printStream = printStream;
32     } // public StructureOutputTextContext(PrintStream printStream) ...
33
34     /**
35      * PrintStream-Instanz, in die die Ausgabe erfolgen soll.
36      */
37     private PrintStream _printStream;
38
39     /**
40      * Gibt die PrintStream-Instanz, in die die Ausgabe erfolgen
41      * soll, zurueck.
42      * @return PrintStream-Instanz, in die die Ausgabe erfolgen soll.
43      */
44     public PrintStream getPrintStream() {
45         return _printStream;
46     } // public PrintStream getPrintStream() ...
47
48     /**
49      * Stack, der StructureOutputTextMethodIntegerSingleton-Instanzen zur Wiederverwendung
            speichert.
50      */
51     private Stack<StructureOutputTextMethodIntegerSingleton> _methodIntegerSingletonStack = new
            Stack<StructureOutputTextMethodIntegerSingleton>();
52
53     /**
54      * Gibt eine zur Verwendung bereite StructureOutputTextMethodIntegerSingleton-Instanz
            zurueck.
55      * @param structure Struktur, mit der die Methode aufgerufen wird.
56      * @return StructureOutputTextMethodIntegerSingleton-Instanz.
57      * @throw IllegalArgumentException structure ist null.
58      */
59     public StructureOutputTextMethodIntegerSingleton getMethodIntegerSingleton(
            IStructureIntegerSingleton structure) {
60         if (structure == null) {
61             throw new IllegalArgumentException("structure = null.");
62         } // if (structure == null) ...
63         StructureOutputTextMethodIntegerSingleton result = _methodIntegerSingletonStack.empty()
                ?
64             new StructureOutputTextMethodIntegerSingleton() : _methodIntegerSingletonStack.pop
                ();
65         result.setStructure(structure);
66         return result;
67     } // public StructureOutputTextMethodIntegerSingleton getMethodIntegerSingleton(
            IStructureIntegerSingleton structure) ...
68
69     /**
70      * Speichert die angegebene Methode zur weiteren Verwendung.
71      * @param method Methode, die gespeichert werden soll.
72      * @throws IllegalArgumentException method ist null.
```

```
73       */
74      public void releaseMethodIntegerSingleton(StructureOutputTextMethodIntegerSingleton method)
             {
75          if (method == null) {
76              throw new IllegalArgumentException("method = null.");
77          } // if (method == null) ...
78          method.reset();
79          _methodIntegerSingletonStack.push(method);
80      } // public void releaseMethodIntegerSingleton(StructureOutputTextMethodIntegerSingleton
            method) ...
81
82      /**
83       * Stack, der StructureOutputTextMethodSingleton-Instanzen zur Wiederverwendung speichert.
84       */
85      private Stack<StructureOutputTextMethodSingleton> _methodSingletonStack = new Stack<
            StructureOutputTextMethodSingleton>();
86
87      /**
88       * Gibt eine zur Verwendung bereite StructureOutputTextMethodSingleton-Instanz zurueck.
89       * @param structure Struktur, mit der die Methode aufgerufen wird.
90       * @return StructureOutputTextMethodSingleton-Instanz.
91       * @throws IllegalArgumentException structure ist null.
92       */
93      public StructureOutputTextMethodSingleton getMethodSingleton(IStructureSingleton structure)
             {
94          if (structure == null) {
95              throw new IllegalArgumentException("structure = null.");
96          } // if (structure == null) ...
97          StructureOutputTextMethodSingleton result = _methodSingletonStack.empty() ?
98              new StructureOutputTextMethodSingleton() : _methodSingletonStack.pop();
99          result.setStructure(structure);
100         return result;
101     } // public StructureOutputTextMethodSingleton getMethodSingleton(IStructureSingleton
            structure) ...
102
103     /**
104      * Speichert die angegebene Methode zur weiteren Verwendung.
105      * @param method Methode, die gespeichert werden soll.
106      * @throws IllegalArgumentException method ist null.
107      */
108     public void releaseMethodSingleton(StructureOutputTextMethodSingleton method) {
109         if (method == null) {
110             throw new IllegalArgumentException("structure = null.");
111         } // if (structure == null) ...
112         method.reset();
113         _methodSingletonStack.push(method);
114     } // public void releaseMethodSingleton(StructureOutputTextMethodSingleton method) ...
115
116     /**
117      * Stack, der StructureOutputTextMethodOne-Instanzen zur Wiederverwendung speichert.
118      */
119     private Stack<StructureOutputTextMethodOne> _methodOneStack = new Stack<
            StructureOutputTextMethodOne>();
120
121     /**
122      * Gibt eine zur Verwendung bereite StructureOutputTextMethodOne-Instanz zurueck.
123      * @param structure Struktur, mit der die Methode aufgerufen wird.
124      * @return StructureOutputTextMethodOne-Instanz.
125      * @throws IllegalArgumentException structure ist null.
126      */
127     public StructureOutputTextMethodOne getMethodOne(IStructureOne structure) {
128         if (structure == null) {
129             throw new IllegalArgumentException("structure = null.");
130         } // if (structure == null) ...
131         StructureOutputTextMethodOne result = _methodOneStack.empty() ?
132             new StructureOutputTextMethodOne() : _methodOneStack.pop();
133         result.setStructure(structure);
134         return result;
135     } // public StructureOutputTextMethodOne getMethodOne(IStructureOne structure) ...
136
137     /**
138      * Speichert die angegebene Methode zur weiteren Verwendung.
139      * @param method Methode, die gespeichert werden soll.
140      * @throws IllegalArgumentException method ist null.
141      */
142     public void releaseMethodOne(StructureOutputTextMethodOne method) {
143         if (method == null) {
144             throw new IllegalArgumentException("structure = null.");
145         } // if (structure == null) ...
146         method.reset();
```

```
147         _methodOneStack.push(method);
148     } // public void releaseMethodSingleton(StructureOutputTextMethodOne method) ...
149 
150     /**
151      * Stack, der StructureOutputTextMethodSum-Instanzen zur Wiederverwendung speichert.
152      */
153     private Stack<StructureOutputTextMethodSum> _methodSumStack = new Stack<
            StructureOutputTextMethodSum>();
154 
155     /**
156      * Gibt eine zur Verwendung bereite StructureOutputTextMethodSum-Instanz zurueck.
157      * @param structure Struktur, mit der die Methode aufgerufen wird.
158      * @return StructureOutputTextMethodSum-Instanz.
159      * @throws IllegalArgumentException structure ist null.
160      */
161     public StructureOutputTextMethodSum getMethodSum(IStructureSum structure) {
162         if (structure == null) {
163             throw new IllegalArgumentException("structure = null.");
164         } // if (structure == null) ...
165         StructureOutputTextMethodSum result = _methodSumStack.empty() ?
166             new StructureOutputTextMethodSum() : _methodSumStack.pop();
167         result.setStructure(structure);
168         return result;
169     } // public StructureOutputTextMethodSum getMethodSum(IStructureSum structure) ...
170 
171     /**
172      * Speichert die angegebene Methode zur weiteren Verwendung.
173      * @param method Methode, die gespeichert werden soll.
174      * @throws IllegalArgumentException method ist null.
175      */
176     public void releaseMethodSum(StructureOutputTextMethodSum method) {
177         if (method == null) {
178             throw new IllegalArgumentException("structure = null.");
179         } // if (structure == null) ...
180         method.reset();
181         _methodSumStack.push(method);
182     } // public void releaseMethodSum(StructureOutputTextMethodSum method) ...
183 
184     /**
185      * Stack, der StructureOutputTextMethodProduct-Instanzen zur Wiederverwendung speichert.
186      */
187     private Stack<StructureOutputTextMethodProduct> _methodProductStack = new Stack<
            StructureOutputTextMethodProduct>();
188 
189     /**
190      * Gibt eine zur Verwendung bereite StructureOutputTextMethodProduct-Instanz zurueck.
191      * @param structure Struktur, mit der die Methode aufgerufen wird.
192      * @return StructureOutputTextMethodProduct-Instanz.
193      * @throws IllegalArgumentException structure ist null.
194      */
195     public StructureOutputTextMethodProduct getMethodProduct(IStructureProduct structure) {
196         if (structure == null) {
197             throw new IllegalArgumentException("structure = null.");
198         } // if (structure == null) ...
199         StructureOutputTextMethodProduct result = _methodProductStack.empty() ?
200             new StructureOutputTextMethodProduct() : _methodProductStack.pop();
201         result.setStructure(structure);
202         return result;
203     } // public StructureOutputTextMethodProduct getMethodProduct(IStructureProduct structure)
            ...
204 
205     /**
206      * Speichert die angegebene Methode zur weiteren Verwendung.
207      * @param method Methode, die gespeichert werden soll.
208      * @throws IllegalArgumentException method ist null.
209      */
210     public void releaseMethodProduct(StructureOutputTextMethodProduct method) {
211         if (method == null) {
212             throw new IllegalArgumentException("structure = null.");
213         } // if (structure == null) ...
214         method.reset();
215         _methodProductStack.push(method);
216     } // public void releaseMethodProduct(StructureOutputTextMethodProduct method) ...
217 
218     /**
219      * Stack, der StructureOutputTextMethodSequence-Instanzen zur Wiederverwendung speichert.
220      */
221     private Stack<StructureOutputTextMethodSequence> _methodSequenceStack = new Stack<
            StructureOutputTextMethodSequence>();
222 
```

```
223     /**
224      * Gibt eine zur Verwendung bereite StructureOutputTextMethodSequence-Instanz zurueck.
225      * @param structure Struktur, mit der die Methode aufgerufen wird.
226      * @return StructureOutputTextMethodSequence-Instanz.
227      * @throws IllegalArgumentException structure ist null.
228      */
229     public StructureOutputTextMethodSequence getMethodSequence(IStructureSequence structure) {
230         if (structure == null) {
231             throw new IllegalArgumentException("structure = null.");
232         } // if (structure == null) ...
233         StructureOutputTextMethodSequence result = _methodSequenceStack.empty() ?
234             new StructureOutputTextMethodSequence() : _methodSequenceStack.pop();
235         result.setStructure(structure);
236         return result;
237     } // public StructureOutputTextMethodSequence getMethodSequence(IStructureSequence
          structure) ...
238
239     /**
240      * Speichert die angegebene Methode zur weiteren Verwendung.
241      * @param method Methode, die gespeichert werden soll.
242      * @throws IllegalArgumentException method ist null.
243      */
244     public void releaseMethodSequence(StructureOutputTextMethodSequence method) {
245         if (method == null) {
246             throw new IllegalArgumentException("structure = null.");
247         } // if (structure == null) ...
248         method.reset();
249         _methodSequenceStack.push(method);
250     } // public void releaseMethodSequence(StructureOutputTextMethodSequence method) ...
251
252     /**
253      * Stack, der StructureOutputTextMethodSet-Instanzen zur Wiederverwendung speichert.
254      */
255     private Stack<StructureOutputTextMethodSet> _methodSetStack = new Stack<
          StructureOutputTextMethodSet>();
256
257     /**
258      * Gibt eine zur Verwendung bereite StructureOutputTextMethodSet-Instanz zurueck.
259      * @param structure Struktur, mit der die Methode aufgerufen wird.
260      * @return StructureOutputTextMethodSet-Instanz.
261      * @throws IllegalArgumentException structure ist null.
262      */
263     public StructureOutputTextMethodSet getMethodSet(IStructureSet structure) {
264         if (structure == null) {
265             throw new IllegalArgumentException("structure = null.");
266         } // if (structure == null) ...
267         StructureOutputTextMethodSet result = _methodSetStack.empty() ?
268             new StructureOutputTextMethodSet() : _methodSetStack.pop();
269         result.setStructure(structure);
270         return result;
271     } // public StructureOutputTextMethodSet getMethodSet(IStructureSet structure) ...
272
273     /**
274      * Speichert die angegebene Methode zur weiteren Verwendung.
275      * @param method Methode, die gespeichert werden soll.
276      * @throws IllegalArgumentException method ist null.
277      */
278     public void releaseMethodSet(StructureOutputTextMethodSet method) {
279         if (method == null) {
280             throw new IllegalArgumentException("structure = null.");
281         } // if (structure == null) ...
282         method.reset();
283         _methodSetStack.push(method);
284     } // public void releaseMethodSet(StructureOutputTextMethodSet method) ...
285
286     /**
287      * Stack, der StructureOutputTextMethodCycle-Instanzen zur Wiederverwendung speichert.
288      */
289     private Stack<StructureOutputTextMethodCycle> _methodCycleStack = new Stack<
          StructureOutputTextMethodCycle>();
290
291     /**
292      * Gibt eine zur Verwendung bereite StructureOutputTextMethodCycle-Instanz zurueck.
293      * @param structure Struktur, mit der die Methode aufgerufen wird.
294      * @return StructureOutputTextMethodCycle-Instanz.
295      * @throws IllegalArgumentException structure ist null.
296      */
297     public StructureOutputTextMethodCycle getMethodCycle(IStructureCycle structure) {
298         if (structure == null) {
299             throw new IllegalArgumentException("structure = null.");
```

```
300         } // if (structure == null) ...
301         StructureOutputTextMethodCycle result = _methodCycleStack.empty() ?
302             new StructureOutputTextMethodCycle() : _methodCycleStack.pop();
303         result.setStructure(structure);
304         return result;
305     } // public StructureOutputTextMethodCycle getMethodCycle(IStructureCycle structure) ...
306
307     /**
308      * Speichert die angegebene Methode zur weiteren Verwendung.
309      * @param method Methode, die gespeichert werden soll.
310      * @throws IllegalArgumentException method ist null.
311      */
312     public void releaseMethodCycle(StructureOutputTextMethodCycle method) {
313         if (method == null) {
314             throw new IllegalArgumentException("structure = null.");
315         } // if (structure == null) ...
316         method.reset();
317         _methodCycleStack.push(method);
318     } // public void releaseMethodCycle(StructureOutputTextMethodCycle method) ...
319
320     /**
321      * Gibt zur angegebenen Strucktur die entsprechende Ausgabemethode zurueck.
322      * @param structure Struktur, zu der die Ausgabemethode zurueckgegeben werden soll.
323      * @return Ausgabemethode zur angegebenen Methode.
324      * @throws IllegalArgumentException Angegebene Struktur ist null oder von
325      *   unbekanntem Typ.
326      */
327     public StructureOutputTextMethod getOutputTextMethod(IStructure structure) {
328         if (structure == null) {
329             throw new IllegalArgumentException("structure = null.");
330         } // if (structure == null) ...
331         if (structure instanceof IStructureIntegerSingleton) {
332             return getMethodIntegerSingleton((IStructureIntegerSingleton) structure);
333         } else if (structure instanceof IStructureSingleton) {
334             return getMethodSingleton((IStructureSingleton) structure);
335         } else if (structure instanceof IStructureOne) {
336             return getMethodOne((IStructureOne) structure);
337         } else if (structure instanceof IStructureSum) {
338             return getMethodSum((IStructureSum) structure);
339         } else if (structure instanceof IStructureProduct) {
340             return getMethodProduct((IStructureProduct) structure);
341         } else if (structure instanceof IStructureSequence) {
342             return getMethodSequence((IStructureSequence) structure);
343         } else if (structure instanceof IStructureSet) {
344             return getMethodSet((IStructureSet) structure);
345         } else if (structure instanceof IStructureCycle) {
346             return getMethodCycle((IStructureCycle) structure);
347         } else {
348             throw new IllegalArgumentException("Struktur von unbekanntem Typ.");
349         }
350     } // public StructureOutputTextMethod getOutputTextMethod(IStructure structure) ...
351
352     /**
353      * Stack der Textausgabe-Methoden, die gerade ausgefuehrt werden.
354      */
355     private Stack<StructureOutputTextMethod> _structureOutputTextMethodStack = new Stack<
         StructureOutputTextMethod>();
356
357     /**
358      * Gibt zurueck, ob der Textausgabe-Methoden-Stack leer ist.
359      * @return Ist der Textausgabe-Methoden-Stack leer?
360      */
361     public boolean isEmpty() {
362         return _structureOutputTextMethodStack.empty();
363     } // public boolean isEmpty() ...
364
365     /**
366      * Entfernt die zuletzt ausgefuehrte Methode.
367      */
368     public void removeMethod() {
369         _structureOutputTextMethodStack.pop();
370     } // public void removeMethod() ...
371
372     /**
373      * Legt die angegebene Methode auf den Methoden-Stack.
374      * @param method Methode, die auf den Methoden-Stack gelegt werden soll.
375      * @throws IllegalArgumentException Die angegebene Methode ist null.
376      */
377     public void pushMethod(StructureOutputTextMethod method) {
378         if (method == null) throw new IllegalArgumentException("method = null.");
```

```
379         _structureOutputTextMethodStack.push(method);
380     } // public void pushMethod(StructureOutputTextMethod method) ...
381 
382     /**
383      * Gibt die aktuelle Methode des Methoden-Stacks zurueck.
384      * @return Aktuelle Methode des Methoden-Stacks.
385      */
386     public StructureOutputTextMethod peekMethod() {
387         return _structureOutputTextMethodStack.peek();
388     } // public StructureOutputTextMethod peekMethod() ...
389 } // class StructureOutputTextContext ...
```

Listing 44: Klasse *at.techmath.boltzmann.output.StructureOutputTextMethod*

```
1  package at.techmath.boltzmann.output;
2
3  import java.io.PrintStream;
4  import at.techmath.boltzmann.sampler.ILabeledStructure;
5  import at.techmath.boltzmann.sampler.ILabel;
6  import at.techmath.boltzmann.sampler.IIntegerLabel;
7  import at.techmath.boltzmann.sampler.IStringLabel;
8
9  /**
10  * Basisklasse der Methoden zur Ausgabe einer Struktur als Text.
11  * @author Stefan Schnabl (e0226245)
12  */
13 abstract class StructureOutputTextMethod {
14     /**
15      * Setzt die Methode in den Anfangszustand zurueck.
16      */
17     public void reset() {
18     } // public void reset() ...
19
20     /**
21      * Fuehrt die aktuelle Methode fort.
22      * @param context StructureOutputTextContext, unter dem die Methode aufgerufen wird.
23      * @throws IllegalArgumentException context ist null.
24      * @thtows IllegalStateException Die Methode wurde in einem ungueltigen Zustand aufgerufen.
25      */
26     public abstract void execute(StructureOutputTextContext context);
27
28     /**
29      * Gibt das Label der angegebenen LabeledStructure-Instanz aus.
30      * @param printStream PrintStream-Instanz, in die geschrieben werden soll.
31      * @param structure Struktur, deren Label ausgegeben werden soll.
32      * @throws IllegalArgumentException printStream oder structure ist null.
33      */
34     protected void outputLabel(PrintStream printStream, ILabeledStructure structure) {
35         if (printStream == null) {
36             throw new IllegalArgumentException("printStream = null.");
37         } // if (printStream == null) ...
38         if (structure == null) {
39             throw new IllegalArgumentException("structure = null.");
40         } // if (structure == null) ...
41         ILabel label = structure.getLabel();
42         if (label instanceof IIntegerLabel) {
43             printStream.print("n");
44             printStream.print(((IIntegerLabel) label).getNumber());
45         } else if (label instanceof IStringLabel) {
46             printStream.print("t\"");
47             printStream.print(((IStringLabel) label).getText().replace('\"', '\''));
48             printStream.print("\"");
49         }
50     } // protected void outputLabel(PrintStream printStream, ILabeledStructure structure) ...
51 } // abstract class StructureOutputTextMethod ...
```

Listing 45: Klasse *at.techmath.boltzmann.output.StructureOutputTextMethodCycle*

```
1  package at.techmath.boltzmann.output;
2
3  import java.io.PrintStream;
4
5  import at.techmath.boltzmann.sampler.IStructureCycle;
6  import at.techmath.boltzmann.sampler.ILabeledStructure;
7
8  /**
9   * Methode, die ein Cycle-Struktur als Text ausgibt.
10  * @author Stefan Schnabl (e0226245)
11  */
12 class StructureOutputTextMethodCycle extends StructureOutputTextMethod {
13     /**
14      * Aktuell auszugebende Struktur.
15      */
16     private IStructureCycle _structure = null;
17
18     /**
19      * Setzt die aktuell auszugebende Struktur.
20      * @param structure Aktuell auszugebende Struktur.
21      */
22     public void setStructure(IStructureCycle structure) {
23         _structure = structure;
24     } // public void setStructure(IStructureCycle structure) ...
25
26     /**
27      * Index des zuletzt ausgegebenen Elements.
28      */
29     private long _currentIndex = -1;
30
31     /**
32      * Setzt die Methode in den Anfangszustand zurueck.
33      */
34     @Override
35     public void reset() {
36         _currentIndex = -1;
37         _structure = null;
38     } // public void reset() ...
39
40     /**
41      * Fuehrt die aktuelle Methode fort.
42      * @param context StructureOutputTextContext, unter dem die Methode aufgerufen wird.
43      * @throws IllegalArgumentException context ist null.
44      * @thtows IllegalStateException Die Methode wurde in einem ungueltigen Zustand aufgerufen.
45      */
46     @Override
47     public void execute(StructureOutputTextContext context) {
48         if (context == null) {
49             throw new IllegalArgumentException("context = null.");
50         } // if (context == null) ...
51         if (_structure == null) {
52             context.releaseMethodCycle(this);
53             context.removeMethod();
54             return;
55         } else { // if (_structure != null) ...
56             long structureLength = _structure.getCount();
57             PrintStream printStream = context.getPrintStream();
58             if (_currentIndex < 0) {
59                 if (structureLength <= 0) {
60                     printStream.print("[]");
61                     if (_structure instanceof ILabeledStructure) { outputLabel(printStream, (
                           ILabeledStructure) _structure); }
62                     context.releaseMethodCycle(this);
63                     context.removeMethod();
64                     return;
65                 } else { // if (structureLength > 0) ...
66                     _currentIndex = 0;
67                     printStream.print("[");
68                     StructureOutputTextMethod method = context.getOutputTextMethod(_structure.
                           getElement(_currentIndex));
69                     context.pushMethod(method);
70                     return;
71                 } // if (structureLength > 0) ...
72             } else if (_currentIndex < structureLength - 1) {
73                 _currentIndex++;
74                 printStream.print(",");
75                 StructureOutputTextMethod method = context.getOutputTextMethod(_structure.
                       getElement(_currentIndex));
76                 context.pushMethod(method);
```

```
77                  return;
78              } else { // if (_currentIndex >= structureLength - 1) ...
79                  printStream.print("]");
80                  if (_structure instanceof ILabeledStructure) { outputLabel(printStream, (
                        ILabeledStructure) _structure); }
81                  context.releaseMethodCycle(this);
82                  context.removeMethod();
83                  return;
84              } // if (_currentIndex >= structureLength - 1) ...
85          } // if (_structure != null) ...
86      } // public void execute(StructureOutputTextContext context) ...
87  } // class StructureOutputTextMethodCycle extends StructureOutputTextMethod ...
```

Listing 46: Klasse *at.techmath.boltzmann.output.StructureOutputTextMethodIntegerSingleton*

```
1  package at.techmath.boltzmann.output;
2
3  import java.io.PrintStream;
4
5  import at.techmath.boltzmann.sampler.IStructureIntegerSingleton;
6  import at.techmath.boltzmann.sampler.ILabeledStructure;
7
8  /**
9   * Methode, die ein Singleton auf einer Teilmenge der natuerlichen Zahlen als Text ausgibt.
10  * @author Stefan Schnabl (e0226245)
11  */
12 class StructureOutputTextMethodIntegerSingleton extends StructureOutputTextMethod {
13     /**
14      * Aktuell auszugebende Struktur.
15      */
16     private IStructureIntegerSingleton _structure = null;
17
18     /**
19      * Setzt die aktuell auszugebende Struktur.
20      * @param structure Aktuell auszugebende Struktur.
21      */
22     public void setStructure(IStructureIntegerSingleton structure) {
23         _structure = structure;
24     } // public void setStructure(IStructureIntegerSingleton structure) ...
25
26     /**
27      * Setzt die Methode in den Anfangszustand zurueck.
28      */
29     @Override
30     public void reset() {
31         _structure = null;
32     } // public void reset() ...
33
34     /**
35      * Fuehrt die aktuelle Methode fort.
36      * @param context StructureOutputTextContext, unter dem die Methode aufgerufen wird.
37      * @throws IllegalArgumentException context ist null.
38      * @thtows IllegalStateException Die Methode wurde in einem ungueltigen Zustand aufgerufen.
39      */
40     @Override
41     public void execute(StructureOutputTextContext context) {
42         if (context == null) {
43             throw new IllegalArgumentException("context = null.");
44         } // if (context == null) ...
45         if (_structure != null) {
46             PrintStream printStream = context.getPrintStream();
47             printStream.print("a");
48             printStream.print(_structure.getNumber());
49             if (_structure instanceof ILabeledStructure) {
50                 outputLabel(printStream, (ILabeledStructure) _structure);
51             } // if (_structure instanceof ILabeledStructure) ...
52         } // if (_structure != null) ...
53         context.releaseMethodIntegerSingleton(this);
54         context.removeMethod();
55     } // public void execute(StructureOutputTextContext context) ...
56 } // class StructureOutputTextMethodIntegerSingleton extends StructureOutputTextMethod ...
```

Listing 47: Klasse *at.techmath.boltzmann.output.StructureOutputTextMethodOne*

```
1  package at.techmath.boltzmann.output;
2
3  import java.io.PrintStream;
4
5  import at.techmath.boltzmann.sampler.IStructureOne;
6  import at.techmath.boltzmann.sampler.ILabeledStructure;
7
8  /**
9   * Methode, die ein One-Struktur als Text ausgibt.
10  * @author Stefan Schnabl (e0226245)
11  */
12 class StructureOutputTextMethodOne extends StructureOutputTextMethod {
13     /**
14      * Aktuell auszugebende Struktur.
15      */
16     private IStructureOne _structure = null;
17
18     /**
19      * Setzt die aktuell auszugebende Struktur.
20      * @param structure Aktuell auszugebende Struktur.
21      */
22     public void setStructure(IStructureOne structure) {
23         _structure = structure;
24     } // public void setStructure(IStructureOne structure) ...
25
26     /**
27      * Setzt die Methode in den Anfangszustand zurueck.
28      */
29     @Override
30     public void reset() {
31         _structure = null;
32     } // public void reset() ...
33
34     /**
35      * Fuehrt die aktuelle Methode fort.
36      * @param context StructureOutputTextContext, unter dem die Methode aufgerufen wird.
37      * @throws IllegalArgumentException context ist null.
38      * @thtows IllegalStateException Die Methode wurde in einem ungueltigen Zustand aufgerufen.
39      */
40     @Override
41     public void execute(StructureOutputTextContext context) {
42         if (context == null) {
43             throw new IllegalArgumentException("context = null.");
44         } // if (context == null) ...
45         if (_structure != null) {
46             PrintStream printStream = context.getPrintStream();
47             printStream.print("o");
48             if (_structure instanceof ILabeledStructure) {
49                 outputLabel(printStream, (ILabeledStructure) _structure);
50             } // if (_structure instanceof ILabeledStructure) ..
51         } // if (_structure != null) ...
52         context.releaseMethodOne(this);
53         context.removeMethod();
54     } // public void execute(StructureOutputTextContext context) ...
55 } // class StructureOutputTextMethodOne extends StructureOutputTextMethod ...
```

Listing 48: Klasse *at.techmath.boltzmann.output.StructureOutputTextMethodProduct*

```
1  package at.techmath.boltzmann.output;
2
3  import java.io.PrintStream;
4  import at.techmath.boltzmann.sampler.IStructureProduct;
5  import at.techmath.boltzmann.sampler.ILabeledStructure;
6
7  /**
8   * Methode, die eine Produkt-Struktur als Text ausgibt.
9   * @author Stefan Schnabl (e0226245)
10  */
11 class StructureOutputTextMethodProduct extends StructureOutputTextMethod {
12     /**
13      * Aktuell auszugebende Struktur.
14      */
15     private IStructureProduct _structure = null;
16
17     /**
18      * Setzt die aktuell auszugebende Struktur.
19      * @param structure Aktuell auszugebende Struktur.
20      */
21     public void setStructure(IStructureProduct structure) {
22         _structure = structure;
23     } // public void setStructure(IStructureProduct structure) ...
24
25     /**
26      * Index des zuletzt ausgegebenen Faktors.
27      */
28     private long _currentIndex = -1;
29
30     /**
31      * Setzt die Methode in den Anfangszustand zurueck.
32      */
33     @Override
34     public void reset() {
35         _structure = null;
36         _currentIndex = -1;
37     } // public void reset() ...
38
39     /**
40      * Fuehrt die aktuelle Methode fort.
41      * @param context StructureOutputTextContext, unter dem die Methode aufgerufen wird.
42      * @throws IllegalArgumentException context ist null.
43      * @thtows IllegalStateException Die Methode wurde in einem ungueltigen Zustand aufgerufen.
44      */
45     @Override
46     public void execute(StructureOutputTextContext context) {
47         if (context == null) {
48             throw new IllegalArgumentException("context = null.");
49         } // if (context == null) ...
50         if (_structure == null) {
51             context.releaseMethodProduct(this);
52             context.removeMethod();
53             return;
54         } else { // if (_structure != null) ...
55             long structureLength = _structure.getCount();
56             PrintStream printStream = context.getPrintStream();
57             if (_currentIndex < 0) {
58                 if (structureLength <= 0) {
59                     printStream.print("<>");
60                     if (_structure instanceof ILabeledStructure) { outputLabel(printStream, (
                           ILabeledStructure) _structure); }
61                     context.releaseMethodProduct(this);
62                     context.removeMethod();
63                     return;
64                 } else { // if (structureLength > 0) ...
65                     _currentIndex = 0;
66                     printStream.print("<");
67                     StructureOutputTextMethod method = context.getOutputTextMethod(_structure.
                           getElement(_currentIndex));
68                     context.pushMethod(method);
69                     return;
70                 } // if (structureLength > 0) ...
71             } else if (_currentIndex < structureLength - 1) {
72                 _currentIndex++;
73                 printStream.print(",");
74                 StructureOutputTextMethod method = context.getOutputTextMethod(_structure.
                       getElement(_currentIndex));
75                 context.pushMethod(method);
76                 return;
```

```
77              } else { // if (_currentIndex >= structureLength - 1) ...
78                  printStream.print(">");
79                  if (_structure instanceof ILabeledStructure) { outputLabel(printStream, (
                        ILabeledStructure) _structure); }
80                  context.releaseMethodProduct(this);
81                  context.removeMethod();
82                  return;
83              } // if (_currentIndex >= structureLength - 1) ...
84          } // if (_structure != null) ...
85      } // public void execute(StructureOutputTextContext context) ...
86  } // class StructureOutputTextMethodProduct extends StructureOutputTextMethod ...
```

Listing 49: Klasse *at.techmath.boltzmann.output.StructureOutputTextMethodSequence*

```
1  package at.techmath.boltzmann.output;
2
3  import java.io.PrintStream;
4
5  import at.techmath.boltzmann.sampler.IStructureSequence;
6  import at.techmath.boltzmann.sampler.ILabeledStructure;
7
8  /**
9   * Methode, die ein Sequence-Struktur als Text ausgibt.
10  * @author Stefan Schnabl (e0226245)
11  */
12 class StructureOutputTextMethodSequence extends StructureOutputTextMethod {
13     /**
14      * Aktuell auszugebende Struktur.
15      */
16     private IStructureSequence _structure = null;
17
18     /**
19      * Setzt die aktuell auszugebende Struktur.
20      * @param structure Aktuell auszugebende Struktur.
21      */
22     public void setStructure(IStructureSequence structure) {
23         _structure = structure;
24     } // public void setStructure(IStructureSequence structure) ...
25
26     /**
27      * Index des zuletzt ausgegebenen Elements.
28      */
29     private long _currentIndex = -1;
30
31     /**
32      * Setzt die Methode in den Anfangszustand zurueck.
33      */
34     @Override
35     public void reset() {
36         _currentIndex = -1;
37         _structure = null;
38     } // public void reset() ...
39
40     /**
41      * Fuehrt die aktuelle Methode fort.
42      * @param context StructureOutputTextContext, unter dem die Methode aufgerufen wird.
43      * @throws IllegalArgumentException context ist null.
44      * @thtows IllegalStateException Die Methode wurde in einem ungueltigen Zustand aufgerufen.
45      */
46     @Override
47     public void execute(StructureOutputTextContext context) {
48         if (context == null) {
49             throw new IllegalArgumentException("context = null.");
50         } // if (context == null) ...
51         if (_structure == null) {
52             context.releaseMethodSequence(this);
53             context.removeMethod();
54             return;
55         } else { // if (_structure != null) ...
56             long structureLength = _structure.getCount();
57             PrintStream printStream = context.getPrintStream();
58             if (_currentIndex < 0) {
59                 if (structureLength <= 0) {
60                     printStream.print("()");
61                     if (_structure instanceof ILabeledStructure) { outputLabel(printStream, (
                           ILabeledStructure) _structure); }
62                     context.releaseMethodSequence(this);
63                     context.removeMethod();
64                     return;
65                 } else { // if (structureLength > 0) ...
66                     _currentIndex = 0;
67                     printStream.print("(");
68                     StructureOutputTextMethod method = context.getOutputTextMethod(_structure.
                           getElement(_currentIndex));
69                     context.pushMethod(method);
70                     return;
71                 } // if (structureLength > 0) ...
72             } else if (_currentIndex < structureLength - 1) {
73                 _currentIndex++;
74                 printStream.print(",");
75                 StructureOutputTextMethod method = context.getOutputTextMethod(_structure.
                       getElement(_currentIndex));
76                 context.pushMethod(method);
```

```
77                  return;
78              } else { // if (_currentIndex >= structureLength - 1) ...
79                  printStream.print(")");
80                  if (_structure instanceof ILabeledStructure) { outputLabel(printStream, (
                        ILabeledStructure) _structure); }
81                  context.releaseMethodSequence(this);
82                  context.removeMethod();
83                  return;
84              } // if (_currentIndex >= structureLength - 1) ...
85          } // if (_structure != null) ...
86      } // public void execute(StructureOutputTextContext context) ...
87  } // class StructureOutputTextMethodSequence extends StructureOutputTextMethod ...
```

Listing 50: Klasse *at.techmath.boltzmann.output.StructureOutputTextMethodSet*

```
1  package at.techmath.boltzmann.output;
2
3  import java.io.PrintStream;
4
5  import at.techmath.boltzmann.sampler.IStructureSet;
6  import at.techmath.boltzmann.sampler.ILabeledStructure;
7
8  /**
9   * Methode, die ein Set-Struktur als Text ausgibt.
10  * @author Stefan Schnabl (e0226245)
11  */
12 class StructureOutputTextMethodSet extends StructureOutputTextMethod {
13     /**
14      * Aktuell auszugebende Struktur.
15      */
16     private IStructureSet _structure = null;
17
18     /**
19      * Setzt die aktuell auszugebende Struktur.
20      * @param structure Aktuell auszugebende Struktur.
21      */
22     public void setStructure(IStructureSet structure) {
23         _structure = structure;
24     } // public void setStructure(IStructureSet structure) ...
25
26     /**
27      * Index des zuletzt ausgegebenen Elements.
28      */
29     private long _currentIndex = -1;
30
31     /**
32      * Setzt die Methode in den Anfangszustand zurueck.
33      */
34     @Override
35     public void reset() {
36         _currentIndex = -1;
37         _structure = null;
38     } // public void reset() ...
39
40     /**
41      * Fuehrt die aktuelle Methode fort.
42      * @param context StructureOutputTextContext, unter dem die Methode aufgerufen wird.
43      * @throws IllegalArgumentException context ist null.
44      * @thtows IllegalStateException Die Methode wurde in einem ungueltigen Zustand aufgerufen.
45      */
46     @Override
47     public void execute(StructureOutputTextContext context) {
48         if (context == null) {
49             throw new IllegalArgumentException("context = null.");
50         } // if (context == null) ...
51         if (_structure == null) {
52             context.releaseMethodSet(this);
53             context.removeMethod();
54             return;
55         } else { // if (_structure != null) ...
56             long structureLength = _structure.getCount();
57             PrintStream printStream = context.getPrintStream();
58             if (_currentIndex < 0) {
59                 if (structureLength <= 0) {
60                     printStream.print("{}");
61                     if (_structure instanceof ILabeledStructure) { outputLabel(printStream, (
                           ILabeledStructure) _structure); }
62                     context.releaseMethodSet(this);
63                     context.removeMethod();
64                     return;
65                 } else { // if (structureLength > 0) ...
66                     _currentIndex = 0;
67                     printStream.print("{");
68                     StructureOutputTextMethod method = context.getOutputTextMethod(_structure.
                           getElement(_currentIndex));
69                     context.pushMethod(method);
70                     return;
71                 } // if (structureLength > 0) ...
72             } else if (_currentIndex < structureLength - 1) {
73                 _currentIndex++;
74                 printStream.print(",");
75                 StructureOutputTextMethod method = context.getOutputTextMethod(_structure.
                       getElement(_currentIndex));
76                 context.pushMethod(method);
```

```
77                  return;
78              } else { // if (_currentIndex >= structureLength - 1) ...
79                  printStream.print("}");
80                  if (_structure instanceof ILabeledStructure) { outputLabel(printStream, (
                        ILabeledStructure) _structure); }
81                  context.releaseMethodSet(this);
82                  context.removeMethod();
83                  return;
84              } // if (_currentIndex >= structureLength - 1) ...
85          } // if (_structure != null) ...
86      } // public void execute(StructureOutputTextContext context) ...
87  } // class StructureOutputTextMethodSet extends StructureOutputTextMethod ...
```

Listing 51: Klasse *at.techmath.boltzmann.output.StructureOutputTextMethodSingleton*

```
1  package at.techmath.boltzmann.output;
2
3  import java.io.PrintStream;
4
5  import at.techmath.boltzmann.sampler.IStructureSingleton;
6  import at.techmath.boltzmann.sampler.ILabeledStructure;
7
8  /**
9   * Methode, die ein Singleton als Text ausgibt.
10  * @author Stefan Schnabl (e0226245)
11  */
12 class StructureOutputTextMethodSingleton extends StructureOutputTextMethod {
13     /**
14      * Aktuell auszugebende Struktur.
15      */
16     private IStructureSingleton _structure = null;
17
18     /**
19      * Setzt die aktuell auszugebende Struktur.
20      * @param structure Aktuell auszugebende Struktur.
21      */
22     public void setStructure(IStructureSingleton structure) {
23         _structure = structure;
24     } // public void setStructure(IStructureSingleton structure) ...
25
26     /**
27      * Setzt die Methode in den Anfangszustand zurueck.
28      */
29     @Override
30     public void reset() {
31         _structure = null;
32     } // public void reset() ...
33
34     /**
35      * Fuehrt die aktuelle Methode fort.
36      * @param context StructureOutputTextContext, unter dem die Methode aufgerufen wird.
37      * @throws IllegalArgumentException context ist null.
38      * @thtows IllegalStateException Die Methode wurde in einem ungueltigen Zustand aufgerufen.
39      */
40     @Override
41     public void execute(StructureOutputTextContext context) {
42         if (context == null) {
43             throw new IllegalArgumentException("context = null.");
44         } // if (context == null) ...
45         if (_structure != null) {
46             PrintStream printStream = context.getPrintStream();
47             printStream.print("a");
48             Object atom = _structure.getAtom();
49             if (atom != null) {
50                 printStream.print(_structure.getAtom().toString());
51             } else { // if (atom == null) ...
52                 printStream.print("null");
53             } // if (atom == null) ...
54             if (_structure instanceof ILabeledStructure) {
55                 outputLabel(printStream, (ILabeledStructure) _structure);
56             } // if (_structure instanceof ILabeledStructure) ...
57         } // if (_structure != null) ...
58         context.releaseMethodSingleton(this);
59         context.removeMethod();
60     } // public void execute(StructureOutputTextContext context) ...
61 } // class StructureOutputTextMethodSingleton extends StructureOutputTextMethod ...
```

Listing 52: Klasse *at.techmath.boltzmann.output.StructureOutputTextMethodSum*

```
1  package at.techmath.boltzmann.output;
2
3  import java.io.PrintStream;
4
5  import at.techmath.boltzmann.sampler.IStructureSum;
6  import at.techmath.boltzmann.sampler.ILabeledStructure;
7
8  /**
9   * Methode, die eine Summen-Struktur als Text ausgibt.
10  * @author Stefan Schnabl (e0226245)
11  */
12 class StructureOutputTextMethodSum extends StructureOutputTextMethod {
13     /**
14      * Aktuell auszugebende Struktur.
15      */
16     private IStructureSum _structure = null;
17
18     /**
19      * Setzt die aktuell auszugebende Struktur.
20      * @param structure Aktuell auszugebende Struktur.
21      */
22     public void setStructure(IStructureSum structure) {
23         _structure = structure;
24     } // public void setStructure(IStructureSum structure) ...
25
26     /**
27      * Wurde die untergeordnete Struktur schon ausgegeben?
28      */
29     boolean _output = false;
30
31     /**
32      * Setzt die Methode in den Anfangszustand zurueck.
33      */
34     @Override
35     public void reset() {
36         _output = false;
37         _structure = null;
38     } // public void reset() ...
39
40     /**
41      * Fuehrt die aktuelle Methode fort.
42      * @param context StructureOutputTextContext, unter dem die Methode aufgerufen wird.
43      * @throws IllegalArgumentException context ist null.
44      * @thtows IllegalStateException Die Methode wurde in einem ungueltigen Zustand aufgerufen.
45      */
46     @Override
47     public void execute(StructureOutputTextContext context) {
48         if (context == null) {
49             throw new IllegalArgumentException("context = null.");
50         } // if (context == null) ...
51         if (_structure == null) {
52             context.releaseMethodSum(this);
53             context.removeMethod();
54             return;
55         } else { // if (_structure != null) ...
56             PrintStream printStream = context.getPrintStream();
57             if (!_output) {
58                 printStream.print("+");
59                 StructureOutputTextMethod method = context.getOutputTextMethod(_structure.
                       getStructure());
60                 context.pushMethod(method);
61                 _output = true;
62                 return;
63             } else { // if (_output) ...
64                 if (_structure instanceof ILabeledStructure) { outputLabel(printStream, (
                       ILabeledStructure) _structure); }
65                 context.releaseMethodSum(this);
66                 context.removeMethod();
67                 return;
68             } // if (_output) ...
69         } // if (_structure != null) ..
70     } // public void execute(StructureOutputTextContext context) ...
71 } // class StructureOutputTextMethodSum extends StructureOutputTextMethod ...
```

Listing 53: Klasse *at.techmath.boltzmann.output.StructureOutputTotalSize*

```
1  package at.techmath.boltzmann.output;
2
3  import java.io.IOException;
4  import java.io.OutputStream;
5  import java.io.PrintStream;
6  import at.techmath.boltzmann.sampler.ISamplingResult;
7
8  /**
9   * Klasse, die die Gesamtgroesse der Strukturen ausgibt.
10  * @author Stefan Schnabl (e0226245)
11  */
12 public class StructureOutputTotalSize implements IStructureOutput {
13     /**
14      * Gibt die angegebene gesampelte Struktur in den angegebenen Stream aus.
15      * @param samplingResult Gesampelte Struktur, die ausgegeben werden soll.
16      * @param outputStream Stream, in den ausgegeben werden soll.
17      * @throws java.io.IOException Fehler bei der Stream-Ausgabe
18      * @throws IllegalArgumentException samplingResult oder outputStream ist null.
19      */
20     public void outputSamplingResult(ISamplingResult samplingResult, OutputStream outputStream)
            throws IOException {
21         if (samplingResult == null) {
22             throw new IllegalArgumentException("samplingResult = null.");
23         } // if (samplingResult == null) ...
24         if (outputStream == null) {
25             throw new IllegalArgumentException("outputStream = null.");
26         } // if (outputStream == null) ...
27         PrintStream printStream = null;
28         try {
29             printStream = new PrintStream(outputStream, false);
30             printStream.println(samplingResult.getSize());
31         } finally {
32             if (printStream != null) {
33                 printStream.flush();
34             } // if (printStream != null) ...
35         }
36     } // public void outputSamplingResult(ISamplingResult samplingResult, OutputStream
           outputStream) throws IOException ...
37 } // public class StructureOutputTotalSize implements IStructureOutput ...
```

A.1.5 Package at.techmath.boltzmann.random

Listing 54: Klasse *at.techmath.boltzmann.random.DefaultGeometricRandomGeneratorFactory*

```
1  package at.techmath.boltzmann.random;
2
3  /**
4   * Standard-Factory fuer Generatoren von
5   * geometrisch verteilten Zufallsvariablen.
6   * @author Stefan Schnabl (e0226245)
7   */
8  public class DefaultGeometricRandomGeneratorFactory implements IGeometricRandomGeneratorFactory
       {
9      /**
10      * gibt einen Generator geometrisch verteilter
11      * Zufallszahlen zuruekc.
12      * @return Generator geometrisch verteilter Zufallszahlen.
13      */
14     public IGeometricRandomGenerator createRandomGenerator() {
15         return new GeometricRandomGeneratorInverseMethod();
16     } // public IGeometricRandomGenerator createRandomGenerator() ...
17 } // public class DefaultGeometricRandomGeneratorFactory implements
       IGeometricRandomGeneratorFactory ...
```

Listing 55: Klasse *at.techmath.boltzmann.random.DefaultLogarithmicRandomGeneratorFactory*

```
1  package at.techmath.boltzmann.random;
2
3  /**
4   * Standard-Factory fuer Generatoren von
5   * logarithmisch verteilten Zufallszahlen.
6   * @author Stefan Schnabl (e0226245)
7   */
8  public class DefaultLogarithmicRandomGeneratorFactory implements
       ILogarithmicRandomGeneratorFactory {
9      /**
10      * Gibt einen Generator logarithmisch verteilter Zufallszahlen zurueck.
11      * @return Generator logarithmisch verteilter Zufallszahlen.
12      */
13     public ILogarithmicRandomGenerator createRandomGenerator() {
14         return new LogarithmicRandomGeneratorKemp();
15     } // public ILogarithmicRandomGenerator createRandomGenerator() ...
16 } // public class DefaultLogarithmicRandomGeneratorFactory implements
       ILogarithmicRandomGeneratorFactory ...
```

Listing 56: Klasse *at.techmath.boltzmann.random.DefaultPoissonRandomGeneratorFactory*

```
1  package at.techmath.boltzmann.random;
2
3  /**
4   * Standard-Factory fuer Generatoren von
5   * Poisson-verteilten Zufallsvariablen.
6   * @author Stefan Schnabl (e0226245)
7   */
8  public class DefaultPoissonRandomGeneratorFactory implements IPoissonRandomGeneratorFactory {
9      /**
10      * Gibt einen Generator Poisson-verteilter
11      * Zufallszahlen zurueck.
12      * @return Generator Poisson-verteilter Zufallszahlen.
13      */
14     public IPoissonRandomGenerator createRandomGenerator() {
15         return new PoissonRandomGeneratorPoissonProcess();
16     } // public IPoissonRandomGenerator createRandomGenerator() ...
17 } // public class DefaultPoissonRandomGeneratorFactory implements IPoissonRandomGenratorFactory
       ...
```

Listing 57: Klasse *at.techmath.boltzmann.random.DefaultUniformRandomGeneratorFactory*

```
1  package at.techmath.boltzmann.random;
2
3  /**
4   * Standard-Factory fuer Generatoren von
5   * auf [0,1) unabhaengig gleichverteilten Zufallszahlen.
6   * @author Stefan Schnabl (e0226245)
7   */
8  public class DefaultUniformRandomGeneratorFactory implements IUniformRandomGeneratorFactory {
9      /**
10      * Gibt einen Zufallszahlengenerator zurueck.
11      * @return Zufallszahlengenerator.
12      */
13     public IUniformRandomGenerator createRandomGenerator() {
14         return new UniformRandomGeneratorComplementaryMultiplyWithCarry();
15     } // public IUniformRandomGenerator createRandomGenerator() ...
16 } // public class DefaultUniformRandomGeneratorFactory implements
      IUniformRandomGeneratorFactory ...
```

Listing 58: Klasse *at.techmath.boltzmann.random.GeometricRandomGeneratorInverseMethod*

```
1  package at.techmath.boltzmann.random;
2
3  /**
4   * Zufallszahlengenerator, der geometrisch verteilte Zufallszahlen mittels
5   * der Inversenmethode erzeugt.
6   * Wenn fuer die Zufallsvariable X fuer alle n aus Nat0 gilt,
7   * dass P(X = n) = x^n * (1-x), dann ist die Verteilungsfunktion
8   * F(n) = P(X <= n) = summe k=0..n x^k * (1-x) = (1-x) * (1-x^(n+1))/(1-x) = 1 - x^(n+1).
9   * Nun gilt fuer die inverse Verteilungsfunktion F^{-1}(p) = min { n aus Nat0 : F(n) >= p } =
10  * min { n aus Nat0 : 1 - x^{n+1} >= p } = min { n aus Nat0 : x^{n+1} <= 1 - p } =
11  * min { n aus Nat0 : (n+1) ln(x) <= ln (1-p) } = min { n aus Nat0 : (n+1) >= ln(1-p) / ln(x) }
        // 0 < x < 1.0 --> ln(x) < 0 =
12  * min { n aus Nat0 : n >= ln(1-p) / ln(x) - 1} = ceiling (ln(1-p) / ln(x)) - 1.
13  * @author Stefan Schnabl (e0226245)
14  */
15 public class GeometricRandomGeneratorInverseMethod implements IGeometricRandomGenerator {
16     /**
17      * Erzeugt eine geometrisch verteilte Zufallsvariable
18      * mit dem angegebenen Parameter.
19      * @param uniformGenerator Zufallszahlengenerator, der auf
20      *   [0,1) unabhaengig gleichverteilte Zufallszahlen erzeugt.
21      * @param parameter Parameter der geometrischen Verteilung.
22      * @return Mit dem angegebenen Parameter geometrisch verteilte Zufallsvariable.
23      * @throws IllegalArgumentException uniformGenerator ist null, parameter <= 0 oder
            parameter >= 1.0D.
24      */
25     public int generateGeometric(IUniformRandomGenerator uniformGenerator, double parameter) {
26         if (uniformGenerator == null) {
27             throw new IllegalArgumentException ("uniformGenerator = null.");
28         } // if (uniformGenerator == null) ...
29         if (parameter <= 0.0D || parameter >= 1.0D) {
30             throw new IllegalArgumentException ("parameter nicht aus (0,1).");
31         } // if (parameter <= 0.0D || parameter >= 1.0D) ...
32         double result = Math.ceil(Math.log(1.0D - uniformGenerator.getDouble()) / Math.log(
               parameter)) - 1.0D;
33         return result < Integer.MAX_VALUE ? (int) result : Integer.MAX_VALUE;
34     } // public int generateGeometric(IUniformRandomGenerator uniformGenerator, double
           parameter) ...
35 } // public class GeometricRandomGeneratorInverseMethod implements IGeometricRandomGenerator
       ...
```

Listing 59: Interface *at.techmath.boltzmann.random.IGeometricRandomGenerator*

```
1  package at.techmath.boltzmann.random;
2
3  /**
4   * Zufallszahlengenerator, der geometrisch verteilte Zufallszahlen erzeugt.
5   * Unter der geomentrischen Verteilung verstehen wir im Kontext der
6   * Boltzmann-Sampler die Verteilung mit Parameter p der Zufallsvariable X,
7   * sodass P(X = n) = p^n * (1 - p) fuer alle n aus Nat0.
8   * @author Stefan Schnabl (e0226245)
9   */
10 public interface IGeometricRandomGenerator {
11     /**
12      * Erzeugt eine geometrisch verteilte Zufallsvariable
13      * mit dem angegebenen Parameter.
14      * @param uniformGenerator Zufallszahlengenerator, der auf
15      *   [0,1) unabhaengig gleichverteilte Zufallszahlen erzeugt.
16      * @param parameter Parameter der geometrischen Verteilung.
17      * @return Mit dem angegebenen Parameter geometrisch verteilte Zufallsvariable.
18      * @throws IllegalArgumentException uniformGenerator ist null, parameter <= 0 oder
            parameter > 1.0D.
19      */
20     int generateGeometric(IUniformRandomGenerator uniformGenerator, double parameter);
21 } // public interface IGeometricRandomGenerator ...
```

Listing 60: Interface *at.techmath.boltzmann.random.IGeometricRandomGeneratorFactory*

```
1  package at.techmath.boltzmann.random;
2
3  /**
4   * Factory, die Generatoren von geometrisch verteilten
5   * Zufallsvariablen erzeugt.
6   * @author Stefan Schnabl (e0226245)
7   */
8  public interface IGeometricRandomGeneratorFactory {
9      /**
10      * Gibt einen Generator geometrisch verteilter
11      * Zufallszahlen zurueck.
12      * @return Generator geometrisch verteilter Zufallszahlen.
13      */
14     IGeometricRandomGenerator createRandomGenerator();
15 } // public interface IGeometricRandomGeneratorFactory ...
```

Listing 61: Interface *at.techmath.boltzmann.random.ILogarithmicRandomGenerator*

```
1  package at.techmath.boltzmann.random;
2
3  /**
4   * Zufallszahlengenerator, der logarithmisch verteilte Zufallszahlen
5   * generieren kann. Eine Zufallsvariable ist logarithmisch verteilt
6   * mit Parameter x, wenn P(X = n) = x^n / n * 1 / ln (1 / (1-x)) fuer alle n aus Nat und 0
        sonst.
7   * @author Stefan Schnabl (e0226245)
8   */
9  public interface ILogarithmicRandomGenerator {
10     /**
11      * Erzeugt eine logarithmisch verteilte Zufallsvariable
12      * mit dem angegebenen Parameter.
13      * @param uniformGenerator Zufallszahlengenerator, der auf
14      *   [0, 1) unabhaengig gleichverteilte Zufallszahlen erzeugt.
15      * @param parameter Parameter der logarithmischen Verteilung.
16      * @return Mit dem angegebenen Parameter logarithmisch verteilte Zufallsvariable.
17      * @throws IllegalArgumentException uniformGenerator ist null oder parameter nicht aus [0,
            1).
18      */
19     int generateLogarithmic(IUniformRandomGenerator uniformGenerator, double parameter);
20 } // public interface ILogarithmicRandomGenerator ...
```

Listing 62: Interface *at.techmath.boltzmann.random.ILogarithmicRandomGeneratorFactory*

```
1  package at.techmath.boltzmann.random;
2
3  /**
4   * Factory, die Generatoren von logarithmisch verteilten
5   * Zufallsvariablen erzeugt.
6   * @author Stefan Schnabl (e0226245)
7   */
8  public interface ILogarithmicRandomGeneratorFactory {
9      /**
10      * Gibt einen Generator logarithmisch verteilter Zufallszahlen zurueck.
11      * @return Generator logarithmisch verteilter Zufallszahlen.
12      */
13     ILogarithmicRandomGenerator createRandomGenerator();
14 } // public interface ILogarithmicRandomGeneratorFactory ...
```

Listing 63: Interface *at.techmath.boltzmann.random.IPoissonRandomGenerator*

```
1  package at.techmath.boltzmann.random;
2
3  /**
4   * Zufallszahlengenerator, der Poisson-verteilte Zufallszahlen
5   * generieren kann.
6   * @author Stefan Schnabl
7   */
8  public interface IPoissonRandomGenerator {
9      /**
10      * Erzeugt eine Poisson-verteilte Zufallsvariable
11      * mit dem angegebenen Parameter.
12      * @param uniformGenerator Zufallszahlengenerator, der auf
13      *   [0,1) unabhaengig gleichverteilte Zufallszahlen erzeugt.
14      * @param parameter Parameter der Poisson-Verteilung.
15      * @return Mit dem angegebenen Parameter Poisson-verteilte Zufallsvariable.
16      * @throws IllegalArgumentException uniformGenerator ist null oder parameter <= 0.0D.
17      */
18     int generatePoisson(IUniformRandomGenerator uniformGenerator, double parameter);
19 } // public interface IPoissonRandomGenerator ...
```

Listing 64: Interface *at.techmath.boltzmann.random.IPoissonRandomGeneratorFactory*

```
1  package at.techmath.boltzmann.random;
2
3  /**
4   * Factory, die Generatoren von Poisson-verteilten
5   * Zufallsvariablen erzeugt.
6   * @author Stefan Schnabl (e0226245)
7   */
8  public interface IPoissonRandomGeneratorFactory {
9      /**
10      * Gibt einen Generator Poisson-verteilter
11      * Zufallszahlen zurueck.
12      * @return Generator Poisson-verteilter Zufallszahlen.
13      */
14     IPoissonRandomGenerator createRandomGenerator();
15 } // public interface IPoissonRandomGeneratorFactory ...
```

Listing 65: Interface *at.techmath.boltzmann.random.IUniformRandomGenerator*

```
1  package at.techmath.boltzmann.random;
2
3  /**
4   * Interface, das einen Zufallszahlengenerator fuer
5   * unabhaengig gleichverteilte Zufallszahlen repraesentiert.
6   * @author Stefan Schnabl (e0226245)
7   */
8  public interface IUniformRandomGenerator {
9      /**
10      * Gibt eine unabhaengig gleichverteilte Zufallszahl zwischen 0 und
11      * Long.MAX_VALUE inklusive zurueck.
12      * @return Unabhaengig gleichverteilte Zufallszahl zwischen 0 und Long.MAX_VALUE inklusive.
13      */
14     long getNonNegativeLong();
15
16     /**
17      * Gibt eine unabhaengig gleichverteilte Zufallszahl zwischen
18      * Long.MIN_VALUE und Long.MAX_VALUE inklusive zurueck.
19      * @return Unabhaengig gleichverteilte Zufallszahl zwischen Long.MIN_VALUE und
20      *   Long.MAX_VALUE inklusive.
21      */
22     long getLong();
23
24     /**
25      * Gibt eine unabhaengig gleichverteilte Zufallszahl zwischen 0 und
26      * Integer.MAX_VALUE inklusive zurueck.
27      * @return Unabhaengig gleichverteilte Zufallszahl zwischen 0 und Integer.MAX_VALUE
            inklusive.
28      */
29     int getNonNegativeInt();
30
31     /**
32      * Gibt eine unabhaengig gleichverteilte Zufallszahl zwischen
33      * Integer.MIN_VALUE und Integer.MAX_VALUE inklusive zurueck.
34      * @return Ungabhaengig gleichverteilte Zufallszahl zwischen Integer.MIN_VALUE und
35      *   Integer.MAX_VALUE inklusive.
36      */
37     int getInt();
38
39     /**
40      * Gibt einen unabhaengig gleichverteilten Wahrheitswert zurueck.
41      * @return Unabhaengig gleichverteilter Wahrheitswert.
42      */
43     boolean getBoolean();
44
45     /**
46      * Gibt eine unabhaengig gleichverteilte Zufallszahl im
47      * Intervall [0, 1) zurueck.
48      * @return Unabhaengig gleichverteilte Zufallszahl im
49      *   Intervall [0, 1).
50      */
51     double getDouble();
52 } // public interface IUniformRandomGenerator ...
```

Listing 66: Interface *at.techmath.boltzmann.random.IUniformRandomGeneratorFactory*

```
1  package at.techmath.boltzmann.random;
2
3  /**
4   * Factory, die Generatoren von auf [0,1) unabhaengig
5   * gleichverteilten Zufallszahlen erzeugt.
6   * @author Stefan Schnabl (e0226245)
7   */
8  public interface IUniformRandomGeneratorFactory {
9      /**
10      * Gibt einen Zufallszahlengenerator zurueck.
11      * @return Zufallszahlengenerator.
12      */
13     IUniformRandomGenerator createRandomGenerator();
14 } // public interface IUniformRandomGeneratorFactory ...
```

Listing 67: Klasse *at.techmath.boltzmann.random.LogarithmicRandomGeneratorKemp*

```
1  package at.techmath.boltzmann.random;
2
3  /**
4   * Zufallszahlengenerator, der nach dem im Paper
5   * "Efficient Generation of Logarithmically Distributed Pseudo-random Variables"
6   * von A.W.Kemp, 1981 angegebenen Algorithmus sampelt.
7   * Algorithmus LB:
8   * 1. h <- ln (1 - parameter)
9   * 2. Generiere zwei unabhaengig auf (0,1] gleichverteilte Zufallsvariablen u1 und u2.
10  * 3. Gib x = floor (1 + ln(u1) / ln(1-exp(u2*h))) zurueck.
11  * @author Stefan Schnabl (e0226245)
12  */
13 public class LogarithmicRandomGeneratorKemp implements ILogarithmicRandomGenerator {
14     /**
15      * Erzeugt eine logarithmisch verteilte Zufallsvariable
16      * mit dem angegebenen Parameter.
17      * @param uniformGenerator Zufallszahlengenerator, der auf
18      *   [0, 1) unabhaengig gleichverteilte Zufallszahlen erzeugt.
19      * @param parameter Parameter der logarithmischen Verteilung.
20      * @return Mit dem angegebenen Parameter logarithmisch verteilte Zufallsvariable.
21      * @throws IllegalArgumentException uniformGenerator ist null oder parameter nicht aus [0,
            1).
22      */
23     public int generateLogarithmic(IUniformRandomGenerator uniformGenerator, double parameter)
           {
24         if (uniformGenerator == null) {
25             throw new IllegalArgumentException("uniformGenerator = null.");
26         } // if (uniformGenerator == null) ...
27         if (parameter < 0.0D || (1.0D - parameter <= 0.0D)) {
28             throw new IllegalArgumentException("parameter nicht aus [0,1)");
29         } // if (parameter <= 0.0D || parameter >= 1.0D) ...
30         double h = Math.log(1.0D - parameter);
31         double u1 = 1.0D - uniformGenerator.getDouble();
32         double u2 = 1.0D - uniformGenerator.getDouble();
33         double result = 1.0D + Math.floor(Math.log(u1) / Math.log(1.0D - Math.exp(u2 * h)));
34         return result > Integer.MAX_VALUE ? Integer.MAX_VALUE : (int) result;
35     } // public int generateLogarithmic(IUniformRandomGenerator uniformGenerator, double
           parameter) ...
36 } // public class LogarithmicRandomGeneratorKemp implements ILogarithmicRandomGenerator ...
```

Listing 68: Klasse *at.techmath.boltzmann.random.PoissonRandomGeneratorPoissonProcess*

```
1  package at.techmath.boltzmann.random;
2
3  /**
4   * Poisson-Zufallszahlen-Generator, der einen Poisson-Prozess simuliert.
5   * Sei X_0, X_1, ... eine Folge unabhaengig exponentialverteilter Zufallszahlen
6   * mit Parameter lambda, dann ist die Zufallsvariable min { n aus Nat0: summe i=0..n X_i > 1 }
7   * Poisson-verteilt mit Parameter lambda. Die Verteilungsfunktion der Exponentialverteilung
8   * ist F(x) = 1 - exp (-lambda * x). Mittels der Inversen-Methode kann daher aus einer
9   * auf (0,1] gleichverteilten Zufallsvariable p durch y = -ln(1-p) / lambda eine
10  * exponentialverteilte Zufallsvariable mit Parameter lambda erzeugt werden.
11  * @author Stefan Schnabl (e0226245)
12  */
13 public class PoissonRandomGeneratorPoissonProcess implements IPoissonRandomGenerator {
14     /**
15      * Erzeugt eine Poisson-verteilte Zufallsvariable
16      * mit dem angegebenen Parameter.
17      * @param uniformGenerator Zufallszahlengenerator, der auf
18      *   [0,1) unabhaengig gleichverteilte Zufallszahlen erzeugt.
19      * @param parameter Parameter der Poisson-Verteilung.
20      * @return Mit dem angegebenen Parameter Poisson-verteilte Zufallsvariable.
21      * @throws IllegalArgumentException uniformGenerator ist null oder parameter <= 0.0D.
22      */
23     public int generatePoisson(IUniformRandomGenerator uniformGenerator, double parameter) {
24         if (uniformGenerator == null) {
25             throw new IllegalArgumentException ("uniformGenerator = null.");
26         } // if (uniformGenerator == null) ...
27         if (parameter <= 0.0D) {
28             throw new IllegalArgumentException ("parameter <= 0");
29         } // if (parameter <= 0.0D) ...
30         double sum = 0.0D;
31         int result = -1;
32         while (sum < parameter) {
33             sum -= Math.log(1.0 - uniformGenerator.getDouble());
34             result++;
35         } // while (sum < parameter) ...
36         return result;
37     } // public int generatePoisson(IUniformRandomGenerator uniformGenerator, double parameter)
           ...
38 } // public class PoissonRandomGeneratorPoissonProcess implements IPoissonRandomGenerator ...
```

Listing 69: Klasse *at.techmath.boltzmann.random.UniformRandomGeneratorComplementaryMultiplyWithCarry*

```
1  package at.techmath.boltzmann.random;
2
3  import java.util.UUID;
4  import java.security.NoSuchAlgorithmException;
5  import java.security.MessageDigest;
6
7  /**
8   * Generator fuer unabhaengig gleichverteilte Zufallszahlen
9   * mittels der Complementary-Multiply-With-Carry-Methode.
10  * @author Stefan Schnabl (e0226245)
11  */
12 public class UniformRandomGeneratorComplementaryMultiplyWithCarry implements
13     IUniformRandomGenerator {
14     /**
15      * Aktueller Zustand des Generators.
16      */
17     private int[] state = new int[4096];
18     private long currentC = 123;
19     private int currentIndex = 4095;
20
21     /**
22      * Erzeugt eine neue Instanz und initialisiert sie mit
23      * einem zufaelligen Seed.
24      */
25     public UniformRandomGeneratorComplementaryMultiplyWithCarry () {
26         initState();
27     } // public UniformRandomGeneratorComplementaryMultiplyWithCarry () ...
28
29     /**
30      * Initialisiert den Zustand auf
31      * einen zufaelligen Seed.
32      */
33     private void initState() {
34         // UUID erzeugen
35         UUID uuid = UUID.randomUUID();
36         // Basis fuer den Hash.
37         // Initialisiert mit UUID und Uhrzeit
38         // ersetzt durch jeweils letzten Hash.
39         byte[] hashSource = new byte[32];
40
41         // hashSource auf UUID und Uhrzeit initialisieren.
42         long[] initValues = new long[4];
43         initValues[0] = uuid.getMostSignificantBits();
44         initValues[1] = uuid.getLeastSignificantBits();
45         initValues[2] = System.currentTimeMillis();
46         initValues[3] = System.nanoTime();
47         int currentHashIndex = 0;
48         for (int c = 0; c < 4; c++) {
49             long currentInitValue = initValues[c];
50             for (int d = 0; d < 8; d++) {
51               hashSource[currentHashIndex++] = (byte) (currentInitValue & 0xFF);
52               currentInitValue >>= 8;
53             } // for (int d = 0; d < 8; d++) ...
54         } // for (int c = 0; c < 4; c++) ...
55
56         try {
57             int currentStateIndex = 0;
58             MessageDigest digest = MessageDigest.getInstance("sha-256");
59             for (int c = 0; c < 512; c++) {
60                 digest.reset();
61                 hashSource = digest.digest(hashSource);
62                 for (int currentHashBase = 0; currentHashBase < 32; currentHashBase += 4) {
63                     int byte1 = hashSource[currentHashBase]; if (byte1 < 0) { byte1 += 256; }
64                     int byte2 = hashSource[currentHashBase + 1]; if (byte2 < 0) { byte2 += 256;
                           }
65                     int byte3 = hashSource[currentHashBase + 2]; if (byte3 < 0) { byte3 += 256;
                           }
66                     int byte4 = hashSource[currentHashBase + 3]; if (byte4 < 0) { byte4 += 256;
                           }
67                     int currentValue = byte1 + (byte2 << 8) + (byte3 << 16) + (byte4 << 24);
68                     state[currentStateIndex++] = currentValue;
69                 } // for (int currentHashBase = 0; currentHashBase < 32; currentHashBase += 4)
                       ...
70             } // for (int c = 0; c < 512; c++) ...
71
72             //for (int c = 0; c < state.length; c++) {
73             //  System.out.print(state[c]); System.out.print(" ");
74             //}
75         } catch (NoSuchAlgorithmException ex) {
```

```
76              System.err.println("Kein SHA-256-Algorithmus in Klassenbibliothek vorhanden!");
77          }
78      } // private void initState() ...
79
80      /**
81       * Gibt eine unabhaengig gleichverteilte Zufallszahl zwischen 0 und
82       * Long.MAX_VALUE inklusive zurueck.
83       * @return Unabhaengig gleichverteilte Zufallszahl zwischen 0 und Long.MAX_VALUE inklusive.
84       */
85      @Override
86      public long getNonNegativeLong() {
87          return getLong() & 0x7fffffffffffffffL;
88      } // public long getNonNegativeLong() ...
89
90      /**
91       * Gibt eine unabhaengig gleichverteilte Zufallszahl zwischen
92       * Long.MIN_VALUE und Long.MAX_VALUE inklusive zurueck.
93       * @return Unabhaengig gleichverteilte Zufallszahl zwischen Long.MIN_VALUE und
94       *   Long.MAX_VALUE inklusive.
95       */
96      @Override
97      public long getLong() {
98          long int1 = getInt(); int1 <<= 32;
99          long int2 = getInt(); if (int2 < 0) { int2 += 0x100000000L; }
100         return int1 + int2;
101     } // public long getLong() ...
102
103     /**
104      * Gibt eine unabhaengig gleichverteilte Zufallszahl zwischen 0 und
105      * Integer.MAX_VALUE inklusive zurueck.
106      * @return Unabhaengig gleichverteilte Zufallszahl zwischen 0 und Integer.MAX_VALUE
           inklusive.
107      */
108     @Override
109     public int getNonNegativeInt() {
110         return getInt() & 0x7fffffff;
111     } // public int getNonNegativeInt() ...
112
113     /**
114      * Gibt eine unabhaengig gleichverteilte Zufallszahl zwischen
115      * Integer.MIN_VALUE und Integer.MAX_VALUE inklusive zurueck.
116      * @return Ungabhaengig gleichverteilte Zufallszahl zwischen Integer.MIN_VALUE und
117      *   Integer.MAX_VALUE inklusive.
118      */
119     @Override
120     public int getInt() {
121         currentIndex = (currentIndex + 1) % 4096;
122         long s = state[currentIndex]; if (s < 0) { s = s + 0x100000000L; }
123         long t = (18782L * s) + currentC;
124         currentC = t >>> 32;
125         long x = (t + currentC) & 0xffffffffL;
126         if (x < currentC) { x++; currentC++; }
127         state[currentIndex] = (int) ((0xfffffffeL - x) & 0xffffffffL);
128         return state[currentIndex];
129     } // public int getInt() ...
130
131     /**
132      * Gibt einen unabhaengig gleichverteilten Wahrheitswert zurueck.
133      * @return Unabhaengig gleichverteilter Wahrheitswert.
134      */
135     @Override
136     public boolean getBoolean() {
137         return getInt() >= 0;
138     } // public boolean getBoolean() ...
139
140     /**
141      * Gibt eine unabhaengig gleichverteilte Zufallszahl im
142      * Intervall [0, 1) zurueck.
143      * @return Unabhaengig gleichverteilte Zufallszahl im
144      *   Intervall [0, 1).
145      */
146     @Override
147     public double getDouble() {
148         long longNumber = getLong();
149         if (longNumber == 0L) {
150             return 0.0D;
151         } else { // if (longNumber != 0L) ...
152             long exponent = 1022L;
153             while (longNumber > 0) {
154                 longNumber <<= 1;
```

```
155                 exponent--;
156             } // while (longNumber > 0) ...
157             longNumber <<= 1;
158             longNumber >>>= 12;
159             return Double.longBitsToDouble(longNumber + (exponent << 52));
160         } // if (longNumber != 0L) ...
161     } // public double getDouble() ...
162 } // public class UniformRandomGeneratorComplementaryMultiplyWithCarry ...
```

A.1.6 Package at.techmath.boltzmann.sampler

Listing 70: Klasse *at.techmath.boltzmann.sampler.ExponentialSampler*

```
1  package at.techmath.boltzmann.sampler;
2
3  import at.techmath.boltzmann.specification.INamedSpecies;
4  import at.techmath.boltzmann.specification.ISpecification;
5  import at.techmath.boltzmann.random.IUniformRandomGenerator;
6  import at.techmath.boltzmann.random.IUniformRandomGeneratorFactory;
7  import at.techmath.boltzmann.random.IGeometricRandomGeneratorFactory;
8  import at.techmath.boltzmann.random.IPoissonRandomGeneratorFactory;
9  import at.techmath.boltzmann.random.ILogarithmicRandomGeneratorFactory;
10 import at.techmath.boltzmann.oracle.IExponentialBoltzmannOracleFactory;
11
12 /**
13  * Standard-Implementierung des IExponentialSampler-Interfaces.
14  * @author Stefan Schnabl (e0226245)
15  */
16 class ExponentialSampler implements IExponentialSampler {
17     /**
18      * Maximal zulaessige Groesse der gesampelten Strukturen.
19      */
20     private long _maxSize = Long.MAX_VALUE;
21
22     /**
23      * Setzt die maximal zulaessige Groesse der gesampelten Strukturen.
24      * Der Standardwert nach Erstellung der Instanz ist Long.MAX_VALUE.
25      * @param maxSize Maximal zulaessige Groesse der gesampelten Strukturen.
26      * @throws IllegalArgumentException maxSize ist kleiner als 0.
27      */
28     public void setMaxSize(long maxSize) {
29         if (maxSize < 0L) {
30             throw new IllegalArgumentException("maxSize < 0.");
31         } // if (maxSize < 0L) ...
32         _maxSize = maxSize;
33     } // public void setMaxSize(long maxSize) ...
34
35     /**
36      * Gibt die maximal zulaessige Groesse der gesampelten Strukturen zurueck.
37      * @return Maximal zulaessige Groesse der gesampelten Strukturen.
38      */
39     public long getMaxSize() {
40         return _maxSize;
41     } // public long getMaxSize() ...
42
43     /**
44      * Factory fuer Generatoren, die auf [0,1) unabhaengig gleichverteilte Zufallsvariablen
            erzeugen.
45      */
46     private IUniformRandomGeneratorFactory _uniformRandomGeneratorFactory;
47
48     /**
49      * Factory fuer Generatoren, die geometrisch verteilte Zufallsvariablen erzeugen.
50      */
51     private IGeometricRandomGeneratorFactory _geometricRandomGeneratorFactory;
52
53     /**
54      * Factory fuer Generatoren, die Poisson-verteilte Zufallsvariablen erzeugen.
55      */
56     private IPoissonRandomGeneratorFactory _poissonRandomGeneratorFactory;
57
58     /**
59      * Factory fuer Generatoren, die logarithmisch verteilte Zufallsvariablen erzeugen.
60      */
61     private ILogarithmicRandomGeneratorFactory _logarithmicRandomGeneratorFactory;
62
63     /**
64      * Factory fuer Boltzmann-Orakel.
65      */
66     private IExponentialBoltzmannOracleFactory _oracleFactory;
67
68     /**
69      * Factory fuer die Grundmenge, auf der die Strukturen erzeugt werden.
70      */
71     private IAtomFactory _atomFactory;
72
73     /**
74      * Spezies, die gesampelt werden soll.
75      */
```

```
76      private INamedSpecies _species;
77
78      /**
79       * Abbildung (Index des Labels) -> Label.
80       */
81      private Label[] _labels = new Label[0];
82
83      /**
84       * Gibt das Label mit dem angegebenen Index zurueck.
85       * @param index Index des gewuenschten Labels.
86       * @return Label mit dem angegebenen Index.
87       * @throws IndexOutOfBoundsException Der Index liegt ausserhalb des gueltigen Bereichs.
88       */
89      protected Label getLabelIndex(int index) {
90          if (index < 0 || index >= _labels.length) {
91              throw new IndexOutOfBoundsException("Index ausserhalb des gueltigen Bereichs.");
92          } // if (index < 0 || index > _labels.length) ...
93          return _labels[index];
94      } // protected Label getLabelIndex(int index) ...
95
96      /**
97       * Abbildung (Index der Spezies) -> Spezies.
98       */
99      private ExponentialSpeciesSampler[] _speciesSampler = new ExponentialSpeciesSampler[0];
100
101     /**
102      * Gibt den Sampler mit dem angegebenen Index zurueck.
103      * @param index Index des gewuenschten Samplers.
104      * @return Sampler mit dem angegebenen Index.
105      * @throws IndexOutOfBoundsException Der Index liegt ausserhalb des gueltigen Bereichs.
106      */
107     protected ExponentialSpeciesSampler getSamplerIndex(int index) {
108         if (index < 0 || index >= _speciesSampler.length) {
109             throw new IndexOutOfBoundsException("Index ausserhalb des gueltigen Bereichs.");
110         } // if (index < 0 || index >= _speciesSampler.length) ...
111         return _speciesSampler[index];
112     } // protected ExponentialSpeciesSampler getSamplerIndex(int index) ...
113
114     /**
115      * Sampelt eine Struktur.
116      * @param x Parameter des Samplers.
117      * @return Struktur, die gesampelt wurde.
118      * @throws at.techmath.boltzmann.sampler.StructureSizeExceededException Wird geworfen,
119      *   wenn die maximal zulaessige Strukturgroesse ueberschritten wurde.
120      */
121     public ISamplingResult sample(double x) throws StructureSizeExceededException {
122         IUniformRandomGenerator uniformRandomGenerator = _uniformRandomGeneratorFactory.
                createRandomGenerator();
123         SamplingContext context = SamplingContext.createContext(
124             _oracleFactory.createOracle(_species.getSpecification()),
125             uniformRandomGenerator,
126             _geometricRandomGeneratorFactory.createRandomGenerator(),
127             _poissonRandomGeneratorFactory.createRandomGenerator(),
128             _logarithmicRandomGeneratorFactory.createRandomGenerator(),
129             x);
130         long maxSize = _maxSize;
131         ExponentialSpeciesSampler sampler = _speciesSampler[_species.getIndex()];
132         SamplerMethod initialMethod = sampler.getMethod();
133         context.pushMethod(initialMethod);
134         while (!context.isEmpty()) {
135             SamplerMethod currentMethod = context.peekMethod();
136             currentMethod.execute(context);
137             if (context.getSize() > maxSize) {
138                 throw new StructureSizeExceededException("Die maximale Groesse fuer gesampelte
                        Strukturen wurde ueberschritten.");
139             } // if (context.getSize() > maxSize) ...
140         } // while (!context.isEmpty()) ...
141         Structure resultStructure = context.getResult();
142
143         SamplingResult result = new SamplingResult(resultStructure, context.getSize());
144
145         if (_atomFactory == null) {
146             // Integer-Labels verteilen
147             IntegerLabelingContext labelingContext = new IntegerLabelingContext(result.getSize
                    (), uniformRandomGenerator);
148             IntegerLabelingMethod initialLabelingMethod = labelingContext.getLabelingMethod(
                    resultStructure);
149             labelingContext.pushMethod(initialLabelingMethod);
150             while (!labelingContext.isEmpty()) {
151                 IntegerLabelingMethod currentMethod = labelingContext.peekMethod();
```

```
152                 currentMethod.execute(labelingContext);
153             } // while (!labelingContext.isEmpty()) ...
154         } else { // if (_atomFactory != null) ...
155             // Labels aus der AtomFactory verteilen
156         } // if (_atomFactory != null) ...
157         //TODO: Labeling der Atome.
158         return result;
159     } // public ISamplingResult sample(double x) ...
160 
161     /**
162      * Sampler fuer Strukturen der Spezies der Charakteristik der leeren Menge.
163      */
164     private ExponentialSpeciesSampler _speciesSamplerOne = new ExponentialSpeciesSamplerOne();
165 
166     /**
167      * Gibt den Sampler fuer Strukturen der Spezies der Charakteristik der leeren Menge zurueck
         .
168      * @return Sampler fuer Strukturen der Spezies der Charakteristik der leeren Menge.
169      */
170     protected ExponentialSpeciesSampler getSpeciesSamplerOne() {
171         return _speciesSamplerOne;
172     } // protected ExponentialSpeciesSampler getSpeciesSamplerOne() ...
173 
174     /**
175      * Sampler fuer Strukturen der Singleton-Spezies.
176      */
177     private ExponentialSpeciesSampler _speciesSamplerSingleton = null;
178 
179     /**
180      * Gibt den Sampler fuer Strukturen der Singleton-Spezies zurueck.
181      * @return Sampler fuer Strukturen der Singleton-Spezies.
182      */
183     protected ExponentialSpeciesSampler getSpeciesSamplerSingleton() {
184         return _speciesSamplerSingleton;
185     } // protected ExponentialSpeciesSampler getSpeciesSamplerSingleton() ...
186 
187     /**
188      * Erzeugt eine neue Instanz anhand der uebergebenen
189      * Spezifikation.
190      * @param uniformRandomGeneratorFactory Factory fuer Generatoren, die auf [0,1) unabhaengig
191      *   gleichverteilte Zufallsvariablen erzeugen.
192      * @param geometricRandomGeneratorFactory Factory fuer Generatoren, die geometrisch
193      *   verteilte Zufallsvariablen erzeugen.
194      * @param poissonRandomGeneratorFactory Factory fuer Generatoren, die Poisson-verteilte
195      *   Zufallsvariablen erzeugen.
196      * @param logarithmicRandomGeneratorFactory Factory fuer Generatoren, die logarithmisch
         verteilte
197      *   Zufallsvariablen erzeugen.
198      * @param oracleFactory Factory fuer Boltzmann-Orakel.
199      * @param atomFactory Factory, die die Grundmenge erzeugt. Wenn dieser Parameter
200      *   null ist, dann wird auf den Teilmengen der natuerlichen Zahlen gesampelt.
201      * @param species Spezies, die gesampelt werden soll.
202      * @return Neue Instanz.
203      * @throws IllegalArgumentException Wird geworfen, wenn eine der RandomGeneratorFactories
         null ist,
204      *   oracleFactory = null, species == null oder species.getSpecfication() == null ist.
205      */
206     public static ExponentialSampler createFromSpecification(IUniformRandomGeneratorFactory
         uniformRandomGeneratorFactory,
207         IGeometricRandomGeneratorFactory geometricRandomGeneratorFactory,
208         IPoissonRandomGeneratorFactory poissonRandomGeneratorFactory,
209         ILogarithmicRandomGeneratorFactory logarithmicRandomGeneratorFactory,
210         IExponentialBoltzmannOracleFactory oracleFactory, IAtomFactory atomFactory,
              INamedSpecies species) {
211         if (uniformRandomGeneratorFactory == null) {
212             throw new IllegalArgumentException("uniformRandomGeneratorFactory = null.");
213         } // if (uniformRandomGeneratorFactory == null) ...
214         if (geometricRandomGeneratorFactory == null) {
215             throw new IllegalArgumentException("geometricRandomGeneratorFactory = null.");
216         } // if (geometricRandomGeneratorFactory == null) ...
217         if (poissonRandomGeneratorFactory == null) {
218             throw new IllegalArgumentException("poissonRandomGeneratorFactory = null.");
219         } // if (poissonRandomGeneratorFactory == null) ...
220         if (logarithmicRandomGeneratorFactory == null) {
221             throw new IllegalArgumentException("logarithmicRandomGeneratorFactory = null.");
222         } // if (logarithmicRandomGeneratorFactory == null) ...
223         if (oracleFactory == null) {
224             throw new IllegalArgumentException("oracleFactory = null.");
225         } // if (oracleFactory == null) ...
226         if (species == null) {
```

```
227             throw new IllegalArgumentException("species = null.");
228         } // if (species == null) ...
229         ISpecification specification = species.getSpecification();
230         if (specification == null) {
231             throw new IllegalArgumentException("species.getSpecification() == null.");
232         } // if (specification == null) ...
233 
234         // Neue Instanz erzeugen
235         ExponentialSampler result = new ExponentialSampler();
236         result._uniformRandomGeneratorFactory = uniformRandomGeneratorFactory;
237         result._geometricRandomGeneratorFactory = geometricRandomGeneratorFactory;
238         result._poissonRandomGeneratorFactory = poissonRandomGeneratorFactory;
239         result._logarithmicRandomGeneratorFactory = logarithmicRandomGeneratorFactory;
240         result._oracleFactory = oracleFactory;
241         result._atomFactory = atomFactory;
242         if (result._atomFactory == null) {
243             result._speciesSamplerSingleton = new ExponentialSpeciesSamplerIntegerSingleton();
244         } else { // if (result._atomFactory != null) ...
245             result._speciesSamplerSingleton = new ExponentialSpeciesSamplerSingleton();
246         } // if (result._atomFactory != null) ...
247         result._species = species;
248 
249         // Labels erzeugen
250         at.techmath.boltzmann.specification.ILabelCollection labelCollection = specification.
                getLabelCollection();
251         if (labelCollection != null) {
252             int count = labelCollection.getCount();
253             result._labels = new Label[count];
254             for (int c = 0; c < count; c++) {
255                 at.techmath.boltzmann.specification.ILabel currentLabel = labelCollection.
                        getLabelIndex(c);
256                 if (currentLabel != null) {
257                     Label newLabel = Label.createFromSpecificationLabel(currentLabel);
258                     result._labels[currentLabel.getIndex()] = newLabel;
259                 } // if (currentLabel != null) ...
260             } // for (int c = 0; c < count; c++) ...
261         } // if (labelCollection == null) ...
262 
263         // Sampler erzeugen
264         {
265             int speciesCount = specification.getCount();
266             result._speciesSampler = new ExponentialSpeciesSampler[speciesCount];
267             for (int c = 0; c < speciesCount; c++) {
268                 at.techmath.boltzmann.specification.INamedSpecies currentSpecies =
                        specification.getSpeciesIndex(c);
269                 if (currentSpecies instanceof at.techmath.boltzmann.specification.ISpeciesSum)
                        {
270                     result._speciesSampler[c] = new ExponentialSpeciesSamplerSum((at.techmath.
                            boltzmann.specification.ISpeciesSum) currentSpecies);
271                 } else if (currentSpecies instanceof at.techmath.boltzmann.specification.
                        ISpeciesProduct) {
272                     result._speciesSampler[c] = new ExponentialSpeciesSamplerProduct((at.
                            techmath.boltzmann.specification.ISpeciesProduct) currentSpecies);
273                 } else if (currentSpecies instanceof at.techmath.boltzmann.specification.
                        ISpeciesSequence) {
274                     result._speciesSampler[c] = new ExponentialSpeciesSamplerSequence((at.
                            techmath.boltzmann.specification.ISpeciesSequence) currentSpecies);
275                 } else if (currentSpecies instanceof at.techmath.boltzmann.specification.
                        ISpeciesSet) {
276                     result._speciesSampler[c] = new ExponentialSpeciesSamplerSet((at.techmath.
                            boltzmann.specification.ISpeciesSet) currentSpecies);
277                 } else if (currentSpecies instanceof at.techmath.boltzmann.specification.
                        ISpeciesCycle) {
278                     result._speciesSampler[c] = new ExponentialSpeciesSamplerCycle((at.techmath
                            .boltzmann.specification.ISpeciesCycle) currentSpecies);
279                 }
280             } // for (int c = 0; c < speciesCount; c++) ...
281         }
282 
283         // Sampler fertig bauen
284         {
285             int speciesCount = result._speciesSampler.length;
286             for (int c = 0; c < speciesCount; c++) {
287                 result._speciesSampler[c].buildInstance(result);
288             } // for (int c = 0; c < speciesCount; c++) ...
289         }
290         return result;
291     } // public static ExponentialSampler createFromSpecification (...) ...
292 } // class ExponentialSampler implements IExponentialSampler ...
```

Listing 71: Klasse *at.techmath.boltzmann.sampler.ExponentialSamplerFactory*

```
1  package at.techmath.boltzmann.sampler;
2
3  import at.techmath.boltzmann.specification.INamedSpecies;
4  import at.techmath.boltzmann.random.IUniformRandomGeneratorFactory;
5  import at.techmath.boltzmann.random.DefaultUniformRandomGeneratorFactory;
6  import at.techmath.boltzmann.random.IGeometricRandomGeneratorFactory;
7  import at.techmath.boltzmann.random.DefaultGeometricRandomGeneratorFactory;
8  import at.techmath.boltzmann.random.IPoissonRandomGeneratorFactory;
9  import at.techmath.boltzmann.random.DefaultPoissonRandomGeneratorFactory;
10 import at.techmath.boltzmann.random.ILogarithmicRandomGeneratorFactory;
11 import at.techmath.boltzmann.random.DefaultLogarithmicRandomGeneratorFactory;
12 import at.techmath.boltzmann.oracle.IExponentialBoltzmannOracleFactory;
13 import at.techmath.boltzmann.oracle.DefaultExponentialBoltzmannOracleFactory;
14
15 /**
16  * Standard-Implementierung des IExponentialSamplerFactory-Interfaces.
17  * @author Stefan Schnabl (e0226245)
18  */
19 public class ExponentialSamplerFactory implements IExponentialSamplerFactory {
20     /**
21      * IUniformRandomGeneratorFactory-Instanz, die verwendet wird.
22      * Wenn null wird eine DefaultUniformRandomGeneratorFactory-Instanz verwendet.
23      */
24     private IUniformRandomGeneratorFactory _uniformRandomGeneratorFactory = null;
25
26     /**
27      * Standard-IUniformRandomGeneratorFactory-Instanz, die verwendet wird,
28      * wenn _uniformRandomGeneratorFactory null ist.
29      */
30     private IUniformRandomGeneratorFactory _defaultUniformRandomGeneratorFactory = new
           DefaultUniformRandomGeneratorFactory();
31
32     /**
33      * Gibt die IUniformRandomGeneratorFactory-Instanz, die verwendet wird, zurueck.
34      * @return IUniformRandomGeneratorFactory-Instanz, die verwendet wird.
35      *   Wenn null zurueckgegeben wird, dann wird eine DefaultUniformRandomGeneratorFactory-
                Instanz verwendet.
36      */
37     public IUniformRandomGeneratorFactory getUniformRandomGeneratorFactory() {
38         return _uniformRandomGeneratorFactory;
39     } // public IUniformRandomGeneratorFactory getUniformRandomGeneratorFactory() ...
40
41     /**
42      * Setzt die IUniformRandomGeneratorFactory-Instanz, die verwendet wird.
43      * @param uniformRandomGeneratorFactory IUniformRandomGeneratorFactory-Instanz, die
                verwendet wird.
44      *   Wenn null angegeben wird, dann wird eine DefaultUniformRandomGeneratorFactory-Instanz
                verwendet.
45      */
46     public void setUniformRandomGeneratorFactory(IUniformRandomGeneratorFactory
           uniformRandomGeneratorFactory) {
47         _uniformRandomGeneratorFactory = uniformRandomGeneratorFactory;
48     } // public void setUniformRandomGeneratorFactory(IUniformRandomGeneratorFactory
           uniformRandomGeneratorFactory) ...
49
50     /**
51      * IGeometricRandomGeneratorFactory-Instanz, die verwendet wird.
52      * Wenn null wird eine DefaultGeometricRandomGeneratorFactory-Instanz verwendet.
53      */
54     private IGeometricRandomGeneratorFactory _geometricRandomGeneratorFactory = null;
55
56     /**
57      * Standard-IGeometricRandomGeneratorFactory-Instanz, die verwendet wird,
58      * wenn _geometricRandomGeneratorFactory null ist.
59      */
60     private IGeometricRandomGeneratorFactory _defaultGeometricRandomGeneratorFactory = new
           DefaultGeometricRandomGeneratorFactory();
61
62     /**
63      * Gibt die IGeometricRandomGeneratorFactory-Instanz, die vewendet wird, zurueck.
64      * @return IGeometricRandomGeneratorFactory-Instanz, die verwendet wird.
65      *   Wenn null zurueckgegeben wird, dann wird eine DefaultGeometricRandomGeneratorFactory-
                Instanz verwendet.
66      */
67     public IGeometricRandomGeneratorFactory getGeometricRandomGeneratorFactory() {
68         return _geometricRandomGeneratorFactory;
69     } // public IGeometricRandomGeneratorFactory getGeometricRandomGeneratorFactory() ...
70
71     /**
```

```
72      * Setzt die IGeometricRandomGeneratorFactory-Instanz, die verwendet wird.
73      * @param geometricRandomGeneratorFactory IGeometricRandomGeneratorFactory-Instanz, die
            verwendet wird.
74      *   Wenn null angegeben wird, dann wird eine DefaultGeometricRandomGeneratorFactory-
            Instanz verwendet.
75      */
76     public void setGeometricRandomGeneratorFactory(IGeometricRandomGeneratorFactory
            geometricRandomGeneratorFactory) {
77         _geometricRandomGeneratorFactory = geometricRandomGeneratorFactory;
78     } // public void setGeometricRandomGeneratorFactory(IGeometricRandomGeneratorFactory
            geometricRandomGeneratorFactory) ...
79
80     /**
81      * IPoissonRandomGeneratorFactory-Instanz, die verwendet wird.
82      * Wenn null wird eine DefaultPoissonRandomGeneratorFactory-Instanz verwendet.
83      */
84     private IPoissonRandomGeneratorFactory _poissonRandomGeneratorFactory = null;
85
86     /**
87      * Standard-IPoissonRandomGeneratorFactory-Instanz, die verwendet wird,
88      * wenn _poissonRandomGeneratorFactory null ist.
89      */
90     private IPoissonRandomGeneratorFactory _defaultPoissonRandomGeneratorFactory = new
            DefaultPoissonRandomGeneratorFactory();
91
92     /**
93      * Gibt die IPoissonRandomGeneratorFactory-Instanz, die verwendet wird, zurueck.
94      * @return IPoissonRandomGeneratorFactory-Instanz, die verwendet wird.
95      *   Wenn null zurueckgegeben wird, dann wird eine DefaultPoissonRandomGeneratorFactory-
            Instanz verwendet.
96      */
97     public IPoissonRandomGeneratorFactory getPoissonRandomGeneratorFactory() {
98         return _poissonRandomGeneratorFactory;
99     } // public IPoissonRandomGeneratorFactory getPoissonRandomGeneratorFactory() ...
100
101    /**
102     * Setzt die IPoissonRandomGeneratorFactory-Instanz, die verwendet wird.
103     * @param poissonRandomGeneratorFactory IPoissonRandomGeneratorFactory-Instanz, die
            verwendet wird.
104     *   Wenn null angegeben wird, dann wird eine DefaultPoissonRandomGeneratorFactory-Instanz
            verwendet.
105     */
106    public void setPoissonRandomGeneratorFactory(IPoissonRandomGeneratorFactory
            poissonRandomGeneratorFactory) {
107        _poissonRandomGeneratorFactory = poissonRandomGeneratorFactory;
108    } // public void setPoissonRandomGeneratorFactory(IPoissonRandomGeneratorFactory
            poissonRandomGeneratorFactory) ...
109
110    /**
111     * ILogarithmicRandomGeneratorFactory-Instanz, die verwendet wird.
112     * Wenn null wird eine DefaultLogarithmicRandomGeneratorFactory-Instanz verwendet.
113     */
114    private ILogarithmicRandomGeneratorFactory _logarithmicRandomGeneratorFactory = null;
115
116    /**
117     * Standard-ILogarithmicRandomGeneratorFactory-Instanz, die verwendet wird,
118     * wenn _logarithmicRandomGeneratorFactory null ist.
119     */
120    private ILogarithmicRandomGeneratorFactory _defaultLogarithmicRandomGeneratorFactory = new
            DefaultLogarithmicRandomGeneratorFactory();
121
122    /**
123     * Gibt die ILogarithmicRandomGeneratorFactory-Instanz, die verwendet wird, zurueck.
124     * @return ILogarithmicRandomGeneratorFactory-Instanz, die verwendet wird.
125     *   Wenn null zurueckgegeben wird, dann wird eine DefaultLogarithmicRandomGeneratorFactory
            -Instanz verwendet.
126     */
127    public ILogarithmicRandomGeneratorFactory getLogarithmicRandomGeneratorFactory() {
128        return _logarithmicRandomGeneratorFactory;
129    } // public ILogarithmicRandomGeneratorFactory getLogarithmicRandomGeneratorFactory() ...
130
131    /**
132     * Setzt die ILogarithmicRandomGeneratorFactory-Instanz, die verwendet wird.
133     * @param logarithmicRandomGeneratorFactory ILogarithmicRandomGeneratorFactory-Instanz, die
            verwendet wird.
134     *   Wenn null angegeben wird, dann wird eine DefaultLogarithmicRandomGeneratorFactory-
            Instanz verwendet.
135     */
136    public void setLogarithmicRandomGeneratorFactory(ILogarithmicRandomGeneratorFactory
            logarithmicRandomGeneratorFactory) {
```

```
137         _logarithmicRandomGeneratorFactory = logarithmicRandomGeneratorFactory;
138     } // public void setLogarithmicRandomGeneratorFactory(ILogarithmicRandomGeneratorFactory
            logarithmicRandomGeneratorFactory) ...
139
140     /**
141      * IExponentialBoltzmannOracleFactory-Instanz, die verwendet wird.
142      * Wenn null wird eine DefaultExponentialBoltzmannOracleFactory-Instanz verwendet.
143      */
144     private IExponentialBoltzmannOracleFactory _exponentialBoltzmannOracleFactory = null;
145
146     /**
147      * Standard-IExponentialBoltzmannOracleFactory-Instanz, die verwendet wird,
148      * wenn _exponentialBoltzmannOracleFactory null ist.
149      */
150     private IExponentialBoltzmannOracleFactory _defaultExponentialBoltzmannOracleFactory = new
            DefaultExponentialBoltzmannOracleFactory();
151
152     /**
153      * Gibt die IExponentialBoltzmannOracleFactory-Instanz, die verwendet wird, zurueck.
154      * @return IExponentialBoltzmannOracleFactory-Instanz, die verwendet wird.
155      *   Wenn null zurueckgegeben wird, dann wird eine DefaultExponentialBoltzmannOracleFactory
                -Instanz verwendet.
156      */
157     public IExponentialBoltzmannOracleFactory getExponentialBoltzmannOracleFactory() {
158         return _exponentialBoltzmannOracleFactory;
159     } // public IExponentialBoltzmannOracleFactory getExponentialBoltzmannOracleFactory() ...
160
161     /**
162      * Setzt die IExponentialBoltzmannOracleFactory-Instanz, die verwendet wird.
163      * @param exponentialBoltzmannOracleFactory IExponentialBoltzmannOracleFactory-Instanz, die
                verwendet wird.
164      *   Wenn null angegeben wird, dann wird eine DefaultExponentialBoltzmannOracleFactory-
                Instanz verwendet.
165      */
166     public void setExponentialBoltzmannOracleFactory(IExponentialBoltzmannOracleFactory
            exponentialBoltzmannOracleFactory) {
167         _exponentialBoltzmannOracleFactory = exponentialBoltzmannOracleFactory;
168     } // public void setExponentialBoltzmannOracleFactory(IExponentialBoltzmannOracleFactory
            exponentialBoltzmannOracleFactory) ...
169
170     /**
171      * Erzeugt fuer die angegebenen Spezies
172      * einen exponentiellen Boltzmann-Sampler.
173      * @param species Spezies, fuer die der Sampler erzeugt werden soll.
174      * @param atomFactory Factory, die die Grundmenge erzeugt. Falls dieser Parameter
175      *   null ist, dann erzeugt der generierte Sampler Atome aus der Menge der natuerlichen
                Zahlen.
176      * @return Exponentieller Boltzmann-Sampler fuer die angegebene Spezies.
177      * @throws IllegalArgumentException Wird geworfen, wenn die
178      *   uebergebene Spezies null ist.
179      */
180     public IExponentialSampler createSampler(INamedSpecies species, IAtomFactory atomFactory) {
181         if (species == null) {
182             throw new IllegalArgumentException("species = null.");
183         } // if (species == null) ...
184         ExponentialSampler result = ExponentialSampler.createFromSpecification(
185             _uniformRandomGeneratorFactory != null ? _uniformRandomGeneratorFactory :
                    _defaultUniformRandomGeneratorFactory,
186             _geometricRandomGeneratorFactory != null ? _geometricRandomGeneratorFactory :
                    _defaultGeometricRandomGeneratorFactory,
187             _poissonRandomGeneratorFactory != null ? _poissonRandomGeneratorFactory :
                    _defaultPoissonRandomGeneratorFactory,
188             _logarithmicRandomGeneratorFactory != null ? _logarithmicRandomGeneratorFactory :
                    _defaultLogarithmicRandomGeneratorFactory,
189             _exponentialBoltzmannOracleFactory != null ? _exponentialBoltzmannOracleFactory :
                    _defaultExponentialBoltzmannOracleFactory,
190             atomFactory,
191             species);
192         return result;
193     } // public IExponentialSampler createSampler(INamedSpecies species, IAtomFactory
            atomFactory) ...
194 } // public class ExponentialSamplerFactory implements IExponentialSamplerFactory ...
```

Listing 72: Klasse *at.techmath.boltzmann.sampler.ExponentialSpeciesSampler*

```
1  package at.techmath.boltzmann.sampler;
2
3  /**
4   * Basisklasse fuer Sampler, die eine konkrete Spezies
5   * sampeln koennen.
6   * @author Stefan Schnabl (e0226245)
7   */
8  abstract class ExponentialSpeciesSampler {
9      /**
10      * Gibt eine Methode zum Sampeln der Spezies zurueck.
11      * @return Methode zum Sampeln der Spezies.
12      */
13     public abstract SamplerMethod getMethod();
14
15     /**
16      * Baut die Instanz zusammen. Es muessen alle referenzierten Sampler existieren.
17      * @throws IllegalArgumentException sampler ist null.
18      */
19     protected abstract void buildInstance(ExponentialSampler sampler);
20
21     /**
22      * Gibt den Wert der erzeugenden Funktion an der Parameter-Stelle zurueck.
23      * @param context Sampling-Context.
24      * @return Wert der erzeugenden Funktion an der Parameter-Stelle.
25      * @throws IllegalArgumentException Wird geworfen, wenn context = null ist.
26      */
27     protected abstract double getGeneratingFunctionValue(SamplingContext context);
28 } // abstract class ExponentialSpeciesSampler ...
```

Listing 73: Klasse *at.techmath.boltzmann.sampler.ExponentialSpeciesSamplerCycle*

```
1  package at.techmath.boltzmann.sampler;
2
3  import java.util.Stack;
4
5  /**
6   * Sampler fuer Strukturen der Zyklen-Spezies.
7   * @author Stefan Schnabl (e0226245)
8   */
9  public class ExponentialSpeciesSamplerCycle extends ExponentialSpeciesSampler {
10     /**
11      * Erzeugt eine neue Instanz.
12      * @param speciesSpecification Spezifikation der Spezies.
13      * @throws IllegalArgumentException speciesSpecification ist null.
14      */
15     public ExponentialSpeciesSamplerCycle(at.techmath.boltzmann.specification.ISpeciesCycle
            speciesSpecification) {
16         super();
17         if (speciesSpecification == null) {
18             throw new IllegalArgumentException("speciesSpecification = null.");
19         } // if (speciesSpecification == null) ...
20         _speciesSpecification = speciesSpecification;
21         _index = speciesSpecification.getIndex();
22     } // public ExponentialSpeciesSamplerCycle(at.techmath.boltzmann.specification.
            ISpeciesCycle speciesSpecification) ...
23
24     /**
25      * Spezifikation der Spezies.
26      */
27     private at.techmath.boltzmann.specification.ISpeciesCycle _speciesSpecification;
28
29     /**
30      * Index der Spezies innerhalb der Spezifikation.
31      */
32     private int _index;
33
34     /**
35      * Gibt den Index der Spezies innerhalb der Spezifikation zurueck.
36      * @return Index der Spezies innerhalb der Spezifikation.
37      */
38     protected int getIndex() {
39         return _index;
40     } // protected int getIndex() ...
41
42     /**
43      * Sampler der Parameter-Spezies.
44      */
45     private ExponentialSpeciesSampler _parameterSampler = null;
46
47     /**
48      * Label, mit dem die Strukturen der Parameter-Spezies versehen sind.
49      */
50     private Label _parameterLabel = null;
51
52     /**
53      * Gibt eine Methode zum Sampeln der Spezies zurueck.
54      * @return Methode zum Sampeln der Spezies.
55      */
56     @Override
57     public SamplerMethod getMethod() {
58         if (_methodStack.empty()) {
59             return new SamplerMethodCycle(this);
60         } else { // if (!_methodStack.empty()) ...
61             return _methodStack.pop();
62         } // if (!_methodStack.empty()) ...
63     } // public SamplerMethod getMethod() ...
64
65     /**
66      * Baut die Instanz zusammen. Es muessen alle referenzierten Sampler existieren.
67      * @throws IllegalArgumentException sampler ist null oder Spezifikation ist ungueltig.
68      */
69     @Override
70     protected void buildInstance(ExponentialSampler sampler) {
71         if (sampler == null) {
72             throw new IllegalArgumentException("sampler = null.");
73         } // if (sampler == null) ...
74         at.techmath.boltzmann.specification.ISpeciesReference speciesReference =
                _speciesSpecification.getSpeciesParameter();
75         if (speciesReference == null) {
```

```
76              throw new IllegalArgumentException("Spezies-Referenz der Parameter-Spezies der
                    Cycle-Spezies ist null.");
77          } // if (speciesReference == null) ...
78          // Spezies-Sampler setzen
79          at.techmath.boltzmann.specification.ISpecies species = speciesReference.getSpecies();
80          if (species == null) {
81              throw new IllegalArgumentException("Spezies-Referenz der Parameter-Spezies der
                    Cycle-Spezies referenziert keine Spezies.");
82          } // if (species == null) ...
83          if (species instanceof at.techmath.boltzmann.specification.ISpeciesOne) {
84              _parameterSampler = sampler.getSpeciesSamplerOne();
85          } else if (species instanceof at.techmath.boltzmann.specification.ISpeciesSingleton) {
86              _parameterSampler = sampler.getSpeciesSamplerSingleton();
87          } else if (species instanceof at.techmath.boltzmann.specification.INamedSpecies) {
88              _parameterSampler = sampler.getSamplerIndex(((at.techmath.boltzmann.specification.
                    INamedSpecies) species).getIndex());
89          } else {
90              throw new IllegalArgumentException("Unbekannter Spezies-Typ.");
91          }
92          // Label setzen
93          at.techmath.boltzmann.specification.ILabel label = speciesReference.getLabel();
94          if (label != null) {
95              _parameterLabel = sampler.getLabelIndex(label.getIndex());
96          } else { // if (label == null) ...
97              _parameterLabel = null;
98          } // if (label == null) ...
99      } // protected void buildInstance(ExponentialSampler sampler) ...
100
101     /**
102      * Gibt den Wert der erzeugenden Funktion an der Parameter-Stelle zurueck.
103      * @param context Sampling-Context.
104      * @return Wert der erzeugenden Funktion an der Parameter-Stelle.
105      * @throws IllegalArgumentException Wird geworfen, wenn context = null ist.
106      */
107     @Override
108     protected double getGeneratingFunctionValue(SamplingContext context) {
109         if (context == null) {
110             throw new IllegalArgumentException("context = null.");
111         } // if (context == null) ...
112         return context.getGeneratingFunctionValue(_index);
113     } // protected double getGeneratingFunctionValue(SamplingContext context) ...
114
115     /**
116      * Stack, der SamplerMethodCycle-Instanzen zur weiteren Verwendung speichert.
117      */
118     private Stack<SamplerMethodCycle> _methodStack = new Stack<SamplerMethodCycle>();
119
120     /**
121      * Speichert die angegebene Methode zur Wiederverwendung.
122      * @param method Methode, die zur Wiederverwendung gespeichert werden soll.
123      * @throws IllegalArgumentException Wird geworfen, wenn method = null ist.
124      */
125     private void pushMethod(SamplerMethodCycle method) {
126         if (method == null) {
127             throw new IllegalArgumentException("method = null.");
128         } // if (method == null) ...
129         method.reset();
130         _methodStack.push(method);
131     } // private void pushMethod(SamplerMethodCycle method) ...
132
133     /**
134      * Sampler-Methode, die eine Struktur der Cycle-Spezies erzeugt.
135      * @author Stefan Schnabl (e0226245)
136      */
137     private class SamplerMethodCycle extends SamplerMethod {
138         /**
139          * Erzeugt eine neue Instanz.
140          * @param parent Sampler, dem die Methode angehoert.
141          * @throws IllegalArgumentException parent = null.
142          */
143         public SamplerMethodCycle(ExponentialSpeciesSamplerCycle parent) {
144             super();
145             if (parent == null) {
146                 throw new IllegalArgumentException("parent = null.");
147             } // if (parent == null) ...
148             _parent = parent;
149         } // public SamplerMethodCycle(ExponentialSpeciesSamplerCycle parent) ...
150
151         /**
152          * Sampler, dem die Methode angehoert.
```

```
153          */
154         private ExponentialSpeciesSamplerCycle _parent;
155 
156         /**
157          * Strukturen der Ergebnis-Struktur.
158          */
159         private Structure[] _resultStructures = null;
160 
161         /**
162          * Index der letzten bereits gesampelten Struktur.
163          * -1 bedeutet, dass noch keine Struktur gesampelt wurde.
164          */
165         private int _currentStructureIndex = -1;
166 
167         /**
168          * Setzt die Methode in den Anfangszustand zurueck.
169          */
170         @Override
171         public void reset() {
172             super.reset();
173             _resultStructures = null;
174             _currentStructureIndex = -1;
175         } // public void reset() ...
176 
177         /**
178          * Fuehrt die aktuelle Methode fort.
179          * @param context Sampling-Context.
180          * @throws IllegalArgumentException Der angegebene Context ist null.
181          * @throws IllegalStateException Die Methode ist in einem ungueltigen Zustand.
182          */
183         @Override
184         public void execute(SamplingContext context) {
185             if (context == null) {
186                 throw new IllegalArgumentException("context = null.");
187             } // if (context == null) ...
188             if (_currentStructureIndex < 0) {
189                 // Ermitteln der Groesse.
190                 int structureSize = context.generateLogarithmicRandomNumber(_parent.
                        _parameterSampler.getGeneratingFunctionValue(context));
191                 // Erzeugen des Ergebnis-Arrays.
192                 _resultStructures = new Structure[structureSize];
193                 if (structureSize <= 0) {
194                     Label label = getLabel();
195                     Structure resultStructure;
196                     if (label != null) {
197                         resultStructure = new LabeledStructureCycle(_resultStructures);
198                         ((LabeledStructureCycle) resultStructure).setLabel(label);
199                     } else { // if (label == null) ...
200                         resultStructure = new StructureCycle(_resultStructures);
201                     } // if (label == null) ...
202                     context.setResult(resultStructure);
203                     context.removeMethod();
204                     _parent.pushMethod(this);
205                     return;
206                 } else { // if (structureSize > 0) ...
207                     _currentStructureIndex = 0;
208                     SamplerMethod firstSampler = _parent._parameterSampler.getMethod();
209                     firstSampler.setLabel(_parent._parameterLabel);
210                     context.pushMethod(firstSampler);
211                     return;
212                 } // if (structureSize > 0) ...
213             } else { // if (_currentStructureIndex >= 0) ...
214                 int structureSize = _resultStructures.length;
215                 if (_currentStructureIndex < structureSize - 1) {
216                     _resultStructures[_currentStructureIndex] = context.getResult();
217                     _currentStructureIndex++;
218                     SamplerMethod nextSampler = _parent._parameterSampler.getMethod();
219                     nextSampler.setLabel(_parent._parameterLabel);
220                     context.pushMethod(nextSampler);
221                     return;
222                 } else { // if (_currentStructureIndex >= structureSize - 1) ...
223                     _resultStructures[_currentStructureIndex] = context.getResult();
224                     Label label = getLabel();
225                     Structure resultStructure;
226                     if (label != null) {
227                         resultStructure = new LabeledStructureCycle(_resultStructures);
228                         ((LabeledStructureCycle) resultStructure).setLabel(label);
229                     } else { // if (label == null) ...
230                         resultStructure = new StructureCycle(_resultStructures);
231                     } // if (label == null) ...
```

```
232                     context.setResult(resultStructure);
233                     context.removeMethod();
234                     _parent.pushMethod(this);
235                     return;
236                 } // if (_currentStructureIndex >= structureSize - 1) ...
237             } // if (_currentStructureIndex >= 0) ...
238         } // public void execute(SamplingContext context) ...
239     } // private class SamplerMethodCycle extends SamplerMethod ...
240 } // public class ExponentialSpeciesSamplerCycle extends ExponentialSpeciesSampler ...
```

Listing 74: Klasse *at.techmath.boltzmann.sampler.ExponentialSpeciesSamplerIntegerSingleton*

```
1  package at.techmath.boltzmann.sampler;
2
3  import java.util.Stack;
4
5  /**
6   * Sampler fuer Strukturen der Singleton-Spezies auf Teilmengen der natuerlichen Zahlen.
7   * @author Stefan Schnabl (e0226245)
8   */
9  class ExponentialSpeciesSamplerIntegerSingleton extends ExponentialSpeciesSampler {
10     /**
11      * Gibt eine Methode zum Sampeln der Spezies zurueck.
12      * @return Methode zum Sampeln der Spezies.
13      */
14     @Override
15     public SamplerMethod getMethod() {
16         if (_methodStack.empty()) {
17             return new SamplerMethodSingleton(this);
18         } else { // if (!_methodStack.empty()) ...
19             return _methodStack.pop();
20         } // if (!_methodStack.empty()) ...
21     } // public SamplerMethod getMethod() ...
22
23     /**
24      * Baut die Instanz zusammen. Es muessen alle referenzierten Sampler existieren.
25      * @throws IllegalArgumentException sampler ist null.
26      */
27     @Override
28     protected void buildInstance(ExponentialSampler sampler) {
29         if (sampler == null) {
30             throw new IllegalArgumentException("sampler = null.");
31         } // if (sampler == null) ...
32     } // protected void buildInstance(ExponentialSampler sampler) ...
33
34     /**
35      * Gibt den Wert der erzeugenden Funktion an der Parameter-Stelle zurueck.
36      * @param context Sampling-Context.
37      * @return Wert der erzeugenden Funktion an der Parameter-Stelle.
38      * @throws IllegalArgumentException Wird geworfen, wenn context = null ist.
39      */
40     @Override
41     protected double getGeneratingFunctionValue(SamplingContext context) {
42         if (context == null) {
43             throw new IllegalArgumentException("context = null.");
44         } // if (context == null) ...
45         return context.getBoltzmannParameter();
46     } // protected double getGeneratingFunctionValue(SamplingContext context) ...
47
48     /**
49      * Stack, der SamplerMethodSingleton-Instanzen zur weiteren Verwendung speichert.
50      */
51     private Stack<SamplerMethodSingleton> _methodStack = new Stack<SamplerMethodSingleton>();
52
53     /**
54      * Speichert die angegebene Methode zur Wiederverwendung.
55      * @param method Methode, die zur Wiederverwendung gespeichert werden soll.
56      * @throws IllegalArgumentException method = null.
57      */
58     private void pushMethod(SamplerMethodSingleton method) {
59         if (method == null) {
60             throw new IllegalArgumentException("method = null.");
61         } // if (method == null) ...
62         method.reset();
63         _methodStack.push(method);
64     } // private void pushMethod(SamplerMethodSingleton method) ...
65
66     /**
67      * Sampler-Methode, die eine Struktur der Singleton-Spezies
68      * auf Teilmengen der natuerlichen Zahlen erzeugt.
69      * @author Stefan Schnabl (e0226245)
70      */
71     private class SamplerMethodSingleton extends SamplerMethod {
72         /**
73          * Erzeugt eine neue Instanz.
74          * @param parent Sampler, dem die Methode angehoert.
75          * @throws IllegalArgumentException parent = null.
76          */
77         public SamplerMethodSingleton(ExponentialSpeciesSamplerIntegerSingleton parent) {
78             super();
79             if (parent == null) {
```

```
80                  throw new IllegalArgumentException("parent = null.");
81              } // if (parent == null) ...
82              _parent = parent;
83          } // public SamplerMethodSingleton(ExponentialSpeciesSamplerIntegerSingleton parent)
                ...
84
85          /**
86           * Sampler, dem die Methode angehoert.
87           */
88          private ExponentialSpeciesSamplerIntegerSingleton _parent;
89
90          /**
91           * Fuehrt die aktuelle Methode fort.
92           * @param context Sampling-Context.
93           * @throws IllegalArgumentException Der angegebene Context ist null.
94           * @throws IllegalStateException Die Methode ist in einem ungueltigen Zustand.
95           */
96          @Override
97          public void execute(SamplingContext context) {
98              if (context == null) {
99                  throw new IllegalArgumentException("context = null.");
100             } // if (context == null) ...
101             Label label = getLabel();
102             StructureIntegerSingleton structure;
103             if (label != null) {
104                 structure = new LabeledStructureIntegerSingleton();
105                 ((LabeledStructureIntegerSingleton) structure).setLabel(label);
106             } else { // if (label == null) ...
107                 structure = new StructureIntegerSingleton();
108             } // if (label == null) ...
109             context.setResult(structure);
110             context.incrementSize();
111             context.removeMethod();
112             _parent.pushMethod(this);
113         } // public void execute(SamplingContext context) ...
114     } // private class SamplerMethodSingleton extends SamplerMethod ...
115 } // class ExponentialSpeciesSamplerIntegerSingleton extends ExponentialSpeciesSampler ...
```

Listing 75: Klasse *at.techmath.boltzmann.sampler.ExponentialSpeciesSamplerOne*

```
1  package at.techmath.boltzmann.sampler;
2
3  import java.util.Stack;
4
5  /**
6   * Sampler fuer Strukturen der Spezies der Charakteristik der leeren Menge.
7   * @author Stefan Schnabl (e0226245)
8   */
9  class ExponentialSpeciesSamplerOne extends ExponentialSpeciesSampler {
10     /**
11      * Gibt eine Methode zum Sampeln der Spezies zurueck.
12      * @return Methode zum Sampeln der Spezies.
13      */
14     @Override
15     public SamplerMethod getMethod() {
16         if (_methodStack.empty()) {
17             return new SamplerMethodOne(this);
18         } else { // if (!_methodStack.empty()) ...
19             return _methodStack.pop();
20         } // if (!_methodStack.empty()) ...
21     } // public SamplerMethod getMethod() ...
22
23     /**
24      * Baut die Instanz zusammen. Es muessen alle referenzierten Sampler existieren.
25      * @throws IllegalArgumentException sampler ist null.
26      */
27     @Override
28     protected void buildInstance(ExponentialSampler sampler) {
29         if (sampler == null) {
30             throw new IllegalArgumentException("sampler = null.");
31         } // if (sampler == null) ...
32     } // protected void buildInstance(ExponentialSampler sampler) ...
33
34     /**
35      * Gibt den Wert der erzeugenden Funktion an der Parameter-Stelle zurueck.
36      * @param context Sampling-Context.
37      * @return Wert der erzeugenden Funktion an der Parameter-Stelle.
38      * @throws IllegalArgumentException Wird geworfen, wenn context = null ist.
39      */
40     @Override
41     protected double getGeneratingFunctionValue(SamplingContext context) {
42         if (context == null) {
43             throw new IllegalArgumentException("context = null.");
44         } // if (context == null) ...
45         return 1.0D;
46     } // protected double getGeneratingFunctionValue(SamplingContext context) ...
47
48     /**
49      * Stack, der SamplerMethodOne-Instanzen zur weiteren Verwendung speichert.
50      */
51     private Stack<SamplerMethodOne> _methodStack = new Stack<SamplerMethodOne>();
52
53     /**
54      * Speichert die angegebene Methode zur Wiederverwendung.
55      * @param method Methode, die zur Wiederverwendung gespeichert werden soll.
56      * @throws IllegalArgumentException method = null.
57      */
58     private void pushMethod(SamplerMethodOne method) {
59         if (method == null) {
60             throw new IllegalArgumentException("method = null.");
61         } // if (method == null) .
62         method.reset();
63         _methodStack.push(method);
64     } // private void pushMethod(SamplerMethodOne method) ...
65
66     /**
67      * Sampler-Methode, die eine Struktur der Spezies der Charakteristik der leeren Menge
           erzeugt.
68      * @author Stefan Schnabl (e0226245)
69      */
70     private class SamplerMethodOne extends SamplerMethod {
71         /**
72          * Erzeugt eine neue Instanz.
73          * @param parent Sampler, dem die Methode angehoert.
74          * @throws IllegalArgumentException parent = null.
75          */
76         public SamplerMethodOne(ExponentialSpeciesSamplerOne parent) {
77             super();
78             if (parent == null) {
```

```
79                  throw new IllegalArgumentException("parent = null.");
80              } // if (parent == null) ...
81              _parent = parent;
82          } // public SamplerMethodOne(ExponentialSpeciesSamplerOne parent) ...
83
84          /**
85           * Sampler, dem die Methode angehoert.
86           */
87          private ExponentialSpeciesSamplerOne _parent;
88
89          /**
90           * Fuehrt die aktuelle Methode fort.
91           * @param context Sampling-Context.
92           * @throws IllegalArgumentException Der angegebene Context ist null.
93           * @throws IllegalStateException Die Methode in einem ungueltigen Zustand.
94           */
95          @Override
96          public void execute(SamplingContext context) {
97              if (context == null) {
98                  throw new IllegalArgumentException("context = null.");
99              } // if (context == null) ...
100             Label label = getLabel();
101             StructureOne structure;
102             if (label != null) {
103                 structure = new LabeledStructureOne();
104                 ((LabeledStructureOne) structure).setLabel(label);
105             } else { // if (label == null) ...
106                 structure = new StructureOne();
107             } // if (label == null) ...
108             context.setResult(structure);
109             context.removeMethod();
110             _parent.pushMethod(this);
111         } // public void execute(SamplingContext context) ...
112     } // private class SamplerMethodOne extends SamplerMethod ...
113 } // class ExponentialSpeciesSamplerOne extends ExponentialSpeciesSampler ...
```

Listing 76: Klasse *at.techmath.boltzmann.sampler.ExponentialSpeciesSamplerProduct*

```
1  package at.techmath.boltzmann.sampler;
2
3  import java.util.Stack;
4
5  /**
6   * Sampler fuer Strukturen der Produkt-Spezies.
7   * @author Stefan Schnabl (e0226245)
8   */
9  class ExponentialSpeciesSamplerProduct extends ExponentialSpeciesSampler {
10     /**
11      * Erzeugt eine neue Instanz.
12      * @param speciesSpecification Spezifikation der Spezies.
13      * @throws IllegalArgumentException speciesSpecification ist null.
14      */
15     public ExponentialSpeciesSamplerProduct(at.techmath.boltzmann.specification.ISpeciesProduct
            speciesSpecification) {
16         super();
17         if (speciesSpecification == null) {
18             throw new IllegalArgumentException("speciesSpecification = null.");
19         } // if (speciesSpecification == null) ...
20         _speciesSpecification = speciesSpecification;
21         _index = speciesSpecification.getIndex();
22     } // public ExponentialSpeciesSamplerProduct(at.techmath.boltzmann.specification.
           ISpeciesProduct speciesSpecification) ...
23
24     /**
25      * Spezifikation der Spezies.
26      */
27     private at.techmath.boltzmann.specification.ISpeciesProduct _speciesSpecification;
28
29     /**
30      * Index der Spezies innerhalb der Spezifikation.
31      */
32     private int _index;
33
34     /**
35      * Gibt den Index der Spezies innerhalb der Spezifikation zurueck.
36      * @return Index der Spezies innerhalb der Spezifikation.
37      */
38     protected int getIndex() {
39         return _index;
40     } // protected int getIndex() ...
41
42     /**
43      * Liste der Sampler, aus denen sich das Produkt zusammensetzt.
44      */
45     private ExponentialSpeciesSampler[] _samplers = null;
46
47     /**
48      * Liste der Labels, mit denen die Faktoren versehen werden.
49      */
50     private Label[] _labels = null;
51
52     /**
53      * Gibt eine Methode zum Sampeln der Spezies zurueck.
54      * @return Methode zum Sampeln der Spezies.
55      */
56     @Override
57     public SamplerMethod getMethod() {
58         if (_methodStack.empty()) {
59             return new SamplerMethodProduct(this);
60         } else { // if (!_methodStack.empty()) ...
61             return _methodStack.pop();
62         } // if (!_methodStack.empty()) ...
63     } // public SamplerMethod getMethod() ...
64
65     /**
66      * Baut die Instanz zusammen. Es muessen alle referenzierten Sampler existieren.
67      * @throws IllegalArgumentException sampler ist null.
68      */
69     @Override
70     protected void buildInstance(ExponentialSampler sampler) {
71         if (sampler == null) {
72             throw new IllegalArgumentException("sampler = null.");
73         } // if (sampler == null) ...
74         int count = _speciesSpecification.getFactorCount();
75         _samplers = new ExponentialSpeciesSampler[count];
76         _labels = new Label[count];
77         for (int c = 0; c < count; c++) {
```

```
78              at.techmath.boltzmann.specification.ISpeciesReference speciesReference =
                    _speciesSpecification.getFactorIndex(c);
79              if (speciesReference == null) {
80                  throw new IllegalArgumentException("Spezies-Referenz eines Faktors der Produkt-
                        Spezies ist null.");
81              } // if (speciesReference == null) ...
82              // Spezies-Sampler setzen.
83              at.techmath.boltzmann.specification.ISpecies species = speciesReference.getSpecies
                    ();
84              if (species == null) {
85                  throw new IllegalArgumentException("Spezies-Referenz eines Faktors der Produkt-
                        Spezies referenziert keine Spezies.");
86              } // if (species == null) ...
87              if (species instanceof at.techmath.boltzmann.specification.ISpeciesOne) {
88                  _samplers[c] = sampler.getSpeciesSamplerOne();
89              } else if (species instanceof at.techmath.boltzmann.specification.ISpeciesSingleton
                    ) {
90                  _samplers[c] = sampler.getSpeciesSamplerSingleton();
91              } else if (species instanceof at.techmath.boltzmann.specification.INamedSpecies) {
92                  _samplers[c] = sampler.getSamplerIndex(((at.techmath.boltzmann.specification.
                        INamedSpecies) species).getIndex());
93              } else {
94                  throw new IllegalArgumentException("Unbekannter Spezies-Typ.");
95              }
96              // Label setzen
97              at.techmath.boltzmann.specification.ILabel label = speciesReference.getLabel();
98              if (label != null) {
99                  _labels[c] = sampler.getLabelIndex(label.getIndex());
100             } else { // if (label == null) ...
101                 _labels[c] = null;
102             } // if (label == null) ...
103         } // for (int c = 0; c < count; c++) ...
104     } // protected void buildInstance(ExponentialSampler sampler) ...
105
106     /**
107      * Gibt den Wert der erzeugenden Funktion an der Parameter-Stelle zurueck.
108      * @param context Sampling-Context.
109      * @return Wert der erzeugenden Funktion an der Parameter-Stelle.
110      * @throws IllegalArgumentException Wird geworfen, wenn context = null ist.
111      */
112     @Override
113     protected double getGeneratingFunctionValue(SamplingContext context) {
114         if (context == null) {
115             throw new IllegalArgumentException("context = null.");
116         } // if (context == null) ...
117         return context.getGeneratingFunctionValue(_index);
118     } // protected double getGeneratingFunctionValue(SamplingContext context) ...
119
120     /**
121      * Stack, der SamplerMethodProduct-Instanzen zur weiteren Verwendung speichert.
122      */
123     private Stack<SamplerMethodProduct> _methodStack = new Stack<SamplerMethodProduct>();
124
125     /**
126      * Speichert die angegebene Methode zur Wiederverwendung.
127      * @param method Methode, die zur Wiederverwendung gespeichert werden soll.
128      * @throws IllegalArgumentException Wird geworfen, wenn method = null ist.
129      */
130     private void pushMethod(SamplerMethodProduct method) {
131         if (method == null) {
132             throw new IllegalArgumentException("method = null.");
133         } // if (method == null) ...
134         method.reset();
135         _methodStack.push(method);
136     } // private void pushMethod(SamplerMethodProduct method) ...
137
138     /**
139      * Sampler-Methode, die eine Struktur der Produkt-Spezies erzeugt.
140      * @author Stefan Schnabl (e0226245)
141      */
142     private class SamplerMethodProduct extends SamplerMethod {
143         /**
144          * Erzeugt eine neue Instanz.
145          * @param parent Sampler, dem die Methode angehoert.
146          * @throws IllegalArgumentException parent = null.
147          */
148         public SamplerMethodProduct(ExponentialSpeciesSamplerProduct parent) {
149             super();
150             if (parent == null) {
151                 throw new IllegalArgumentException("parent = null.");
```

```
152             } // if (parent == null) ...
153             _parent = parent;
154         } // public SamplerMethodProduct(ExponentialSpeciesSamplerProduct parent) ...
155 
156         /**
157          * Sampler, dem die Methode angehoert.
158          */
159         private ExponentialSpeciesSamplerProduct _parent;
160 
161         /**
162          * Strukturen der Ergebnis-Struktur.
163          */
164         private Structure[] _resultStructures = null;
165 
166         /**
167          * Index der letzten bereits gesampelten Struktur.
168          * -1 bedeutet, dass noch keine Struktur gesampelt wurde.
169          */
170         private int _currentStructureIndex = -1;
171 
172         /**
173          * Setzt die Methode in den Anfangszustand zurueck.
174          */
175         @Override
176         public void reset() {
177             super.reset();
178             _resultStructures = null;
179             _currentStructureIndex = -1;
180         } // public void reset() ...
181 
182         /**
183          * Fuehrt die aktuelle Methode fort.
184          * @param context Sampling-Context.
185          * @throws IllegalArgumentException Der angegebene Context ist null.
186          * @throws IllegalStateException Die Methode ist in einem ungueltigen Zustand.
187          */
188         @Override
189         public void execute(SamplingContext context) {
190             if (context == null) {
191                 throw new IllegalArgumentException("context = null.");
192             } // if (context == null) ...
193             int factorCount = _parent._samplers.length;
194             if (_currentStructureIndex < 0) {
195                 _resultStructures = new Structure[factorCount];
196                 _currentStructureIndex = 0;
197                 SamplerMethod firstSampler = _parent._samplers[_currentStructureIndex].
                        getMethod();
198                 firstSampler.setLabel(_parent._labels[_currentStructureIndex]);
199                 context.pushMethod(firstSampler);
200                 return;
201             } else if (_currentStructureIndex < factorCount - 1) {
202                 _resultStructures[_currentStructureIndex] = context.getResult();
203                 _currentStructureIndex++;
204                 SamplerMethod nextSampler = _parent._samplers[_currentStructureIndex].getMethod
                        ();
205                 nextSampler.setLabel(_parent._labels[_currentStructureIndex]);
206                 context.pushMethod(nextSampler);
207                 return;
208             } else { // if (_currentStructureIndex >= factorCount - 1) ...
209                 _resultStructures[_currentStructureIndex] = context.getResult();
210                 Label label = getLabel();
211                 Structure resultStructure;
212                 if (label != null) {
213                     resultStructure = new LabeledStructureProduct(_resultStructures);
214                     ((LabeledStructureProduct) resultStructure).setLabel(label);
215                 } else { // if (label == null) ...
216                     resultStructure = new StructureProduct(_resultStructures);
217                 } // if (label == null) ...
218                 context.setResult(resultStructure);
219                 context.removeMethod();
220                 _parent.pushMethod(this);
221                 return;
222             } // if (_currentStructureIndex >= factorCount - 1) ...
223         } // public void execute(SamplingContext context) ...
224     } // private class SamplerMethodProduct extends SamplerMethod ...
225 } // class ExponentialSpeciesSamplerProduct extends ExponentialSpeciesSampler ...
```

Listing 77: Klasse *at.techmath.boltzmann.sampler.ExponentialSpeciesSamplerSequence*

```
1  package at.techmath.boltzmann.sampler;
2
3  import java.util.Stack;
4
5  /**
6   * Sampler fuer Strukturen der Sequence-Spezies.
7   * @author Stefan Schnabl (e0226245)
8   */
9  public class ExponentialSpeciesSamplerSequence extends ExponentialSpeciesSampler {
10     /**
11      * Erzeugt eine neue Instanz.
12      * @param speciesSpecification Spezifikation der Spezies.
13      * @throws IllegalArgumentException speciesSpecification ist null.
14      */
15     public ExponentialSpeciesSamplerSequence(at.techmath.boltzmann.specification.
            ISpeciesSequence speciesSpecification) {
16         super();
17         if (speciesSpecification == null) {
18             throw new IllegalArgumentException("speciesSpecification = null.");
19         } // if (speciesSpecification == null) ...
20         _speciesSpecification = speciesSpecification;
21         _index = speciesSpecification.getIndex();
22     } // public ExponentialSpeciesSamplerSequence(at.techmath.boltzmann.specification.
            ISpeciesSequence speciesSpecification) ...
23
24     /**
25      * Spezifikation der Spezies.
26      */
27     private at.techmath.boltzmann.specification.ISpeciesSequence _speciesSpecification;
28
29     /**
30      * Index der Spezies innerhalb der Spezifikation.
31      */
32     private int _index;
33
34     /**
35      * Gibt den Index der Spezies innerhalb der Spezifikation zurueck.
36      * @return Index der Spezies innerhalb der Spezifikation.
37      */
38     protected int getIndex() {
39         return _index;
40     } // protected int getIndex() ...
41
42     /**
43      * Sampler der Parameter-Spezies.
44      */
45     private ExponentialSpeciesSampler _parameterSampler = null;
46
47     /**
48      * Label, mit dem die Strukturen der Parameter-Spezies versehen sind.
49      */
50     private Label _parameterLabel = null;
51
52     /**
53      * Gibt eine Methode zum Sampeln der Spezies zurueck.
54      * @return Methode zum Sampeln der Spezies.
55      */
56     @Override
57     public SamplerMethod getMethod() {
58         if (_methodStack.empty()) {
59             return new SamplerMethodSequence(this);
60         } else { // if (!_methodStack.empty()) ...
61             return _methodStack.pop();
62         } // if (!_methodStack.empty()) ...
63     } // public SamplerMethod getMethod() ...
64
65     /**
66      * Baut die Instanz zusammen. Es muessen alle referenzierten Sampler existieren.
67      * @throws IllegalArgumentException sampler ist null oder Spezifikation ist ungueltig.
68      */
69     @Override
70     protected void buildInstance(ExponentialSampler sampler) {
71         if (sampler == null) {
72             throw new IllegalArgumentException("sampler = null.");
73         } // if (sampler == null) ...
74         at.techmath.boltzmann.specification.ISpeciesReference speciesReference =
                _speciesSpecification.getSpeciesParameter();
75         if (speciesReference == null) {
```

```
76              throw new IllegalArgumentException("Spezies-Referenz der Parameter-Spezies der
                    Sequence-Spezies ist null.");
77          } // if (speciesReference == null) ...
78          // Spezies-Sampler setzen
79          at.techmath.boltzmann.specification.ISpecies species = speciesReference.getSpecies();
80          if (species == null) {
81              throw new IllegalArgumentException("Spezies-Referenz der Parameter-Spezies der
                    Sequence-Spezies referenziert keine Spezies.");
82          } // if (species == null) ...
83          if (species instanceof at.techmath.boltzmann.specification.ISpeciesOne) {
84              _parameterSampler = sampler.getSpeciesSamplerOne();
85          } else if (species instanceof at.techmath.boltzmann.specification.ISpeciesSingleton) {
86              _parameterSampler = sampler.getSpeciesSamplerSingleton();
87          } else if (species instanceof at.techmath.boltzmann.specification.INamedSpecies) {
88              _parameterSampler = sampler.getSamplerIndex(((at.techmath.boltzmann.specification.
                    INamedSpecies) species).getIndex());
89          } else {
90              throw new IllegalArgumentException("Unbekannter Spezies-Typ.");
91          }
92          // Label setzen
93          at.techmath.boltzmann.specification.ILabel label = speciesReference.getLabel();
94          if (label != null) {
95              _parameterLabel = sampler.getLabelIndex(label.getIndex());
96          } else { // if (label == null) ...
97              _parameterLabel = null;
98          } // if (label == null) ...
99      } // protected void buildInstance(ExponentialSampler sampler) ...
100
101     /**
102      * Gibt den Wert der erzeugenden Funktion an der Parameter-Stelle zurueck.
103      * @param context Sampling-Context.
104      * @return Wert der erzeugenden Funktion an der Parameter-Stelle.
105      * @throws IllegalArgumentException Wird geworfen, wenn context = null ist.
106      */
107     @Override
108     protected double getGeneratingFunctionValue(SamplingContext context) {
109         if (context == null) {
110             throw new IllegalArgumentException("context = null.");
111         } // if (context == null) ...
112         return context.getGeneratingFunctionValue(_index);
113     } // protected double getGeneratingFunctionValue(SamplingContext context) ...
114
115     /**
116      * Stack, der SamplerMethodSequence-Instanzen zur weiteren Verwendung speichert.
117      */
118     private Stack<SamplerMethodSequence> _methodStack = new Stack<SamplerMethodSequence>();
119
120     /**
121      * Speichert die angegebene Methode zur Wiederverwendung.
122      * @param method Methode, die zur Wiederverwendung gespeichert werden soll.
123      * @throws IllegalArgumentException Wird geworfen, wenn method = null ist.
124      */
125     private void pushMethod(SamplerMethodSequence method) {
126         if (method == null) {
127             throw new IllegalArgumentException("method = null.");
128         } // if (method == null) ...
129         method.reset();
130         _methodStack.push(method);
131     } // private void pushMethod(SamplerMethodSequence method) ...
132
133     /**
134      * Sampler-Methode, die eine Struktur der Sequence-Spezies erzeugt.
135      * @author Stefan Schnabl (e0226245)
136      */
137     private class SamplerMethodSequence extends SamplerMethod {
138         /**
139          * Erzeugt eine neue Instanz.
140          * @param parent Sampler, dem die Methode angehoert.
141          * @throws IllegalArgumentException parent = null.
142          */
143         public SamplerMethodSequence(ExponentialSpeciesSamplerSequence parent) {
144             super();
145             if (parent == null) {
146                 throw new IllegalArgumentException("parent = null.");
147             } // if (parent == null) ...
148             _parent = parent;
149         } // public SamplerMethodSequence(ExponentialSpeciesSamplerSequence parent) ...
150
151         /**
152          * Sampler, dem die Methode angehoert.
```

```
153          */
154         private ExponentialSpeciesSamplerSequence _parent;
155 
156         /**
157          * Strukturen der Ergebnis-Struktur.
158          */
159         private Structure[] _resultStructures = null;
160 
161         /**
162          * Index der letzten bereits gesampelten Struktur.
163          * -1 bedeutet, dass noch keine Struktur gesampelt wurde.
164          */
165         private int _currentStructureIndex = -1;
166 
167         /**
168          * Setzt die Methode in den Anfangszustand zurueck.
169          */
170         @Override
171         public void reset() {
172             super.reset();
173             _resultStructures = null;
174             _currentStructureIndex = -1;
175         } // public void reset() ...
176 
177         /**
178          * Fuehrt die aktuelle Methode fort.
179          * @param context Sampling-Context.
180          * @throws IllegalArgumentException Der angegebene Context ist null.
181          * @throws IllegalStateException Die Methode ist in einem ungueltigen Zustand.
182          */
183         @Override
184         public void execute(SamplingContext context) {
185             if (context == null) {
186                 throw new IllegalArgumentException("context = null.");
187             } // if (context == null) ...
188             if (_currentStructureIndex < 0) {
189                 // Ermitteln der Groesse.
190                 int structureSize = context.generateGeometricRandomNumber(_parent.
                        _parameterSampler.getGeneratingFunctionValue(context));
191                 // Erzeugen des Ergebnis-Arrays.
192                 _resultStructures = new Structure[structureSize];
193                 if (structureSize <= 0) {
194                     Label label = getLabel();
195                     Structure resultStructure;
196                     if (label != null) {
197                         resultStructure = new LabeledStructureSequence(_resultStructures);
198                         ((LabeledStructureSequence) resultStructure).setLabel(label);
199                     } else { // if (label == null) ...
200                         resultStructure = new StructureSequence(_resultStructures);
201                     } // if (label == null) ...
202                     context.setResult(resultStructure);
203                     context.removeMethod();
204                     _parent.pushMethod(this);
205                     return;
206                 } else { // if (structureSize > 0) ...
207                     _currentStructureIndex = 0;
208                     SamplerMethod firstSampler = _parent._parameterSampler.getMethod();
209                     firstSampler.setLabel(_parent._parameterLabel);
210                     context.pushMethod(firstSampler);
211                     return;
212                 } // if (structureSize > 0) ...
213             } else { // if (_currentStructureIndex >= 0) ...
214                 int structureSize = _resultStructures.length;
215                 if (_currentStructureIndex < structureSize - 1) {
216                     _resultStructures[_currentStructureIndex] = context.getResult();
217                     _currentStructureIndex++;
218                     SamplerMethod nextSampler = _parent._parameterSampler.getMethod();
219                     nextSampler.setLabel(_parent._parameterLabel);
220                     context.pushMethod(nextSampler);
221                     return;
222                 } else { // if (_currentStructureIndex >= structureSize - 1) ...
223                     _resultStructures[_currentStructureIndex] = context.getResult();
224                     Label label = getLabel();
225                     Structure resultStructure;
226                     if (label != null) {
227                         resultStructure = new LabeledStructureSequence(_resultStructures);
228                         ((LabeledStructureSequence) resultStructure).setLabel(label);
229                     } else { // if (label == null) ...
230                         resultStructure = new StructureSequence(_resultStructures);
231                     } // if (label == null) ...
```

```
232                     context.setResult(resultStructure);
233                     context.removeMethod();
234                     _parent.pushMethod(this);
235                     return;
236                 } // if (_currentStructureIndex >= structureSize - 1) ...
237             } // if (_currentStructureIndex >= 0) ...
238         } // public void execute(SamplingContext context) ...
239     } // private class SamplerMethodSequence extends SamplerMethod ...
240 } // public class ExponentialSpeciesSamplerSequence extends ExponentialSpeciesSampler ...
```

Listing 78: Klasse *at.techmath.boltzmann.sampler.ExponentialSpeciesSamplerSet*

```
1  package at.techmath.boltzmann.sampler;
2
3  import java.util.Stack;
4
5  /**
6   * Sampler fuer Strukturen der Set-Spezies.
7   * @author Stefan Schnabl (e0226245)
8   */
9  public class ExponentialSpeciesSamplerSet extends ExponentialSpeciesSampler {
10     /**
11      * Erzeugt eine neue Instanz.
12      * @param speciesSpecification Spezifikation der Spezies.
13      * @throws IllegalArgumentException speciesSpecification ist null.
14      */
15     public ExponentialSpeciesSamplerSet(at.techmath.boltzmann.specification.ISpeciesSet
            speciesSpecification) {
16         super();
17         if (speciesSpecification == null) {
18             throw new IllegalArgumentException("speciesSpecification = null.");
19         } // if (speciesSpecification == null) ...
20         _speciesSpecification = speciesSpecification;
21         _index = speciesSpecification.getIndex();
22     } // public ExponentialSpeciesSamplerSet(at.techmath.boltzmann.specification.ISpeciesSet
            speciesSpecification) ...
23
24     /**
25      * Spezifikation der Spezies.
26      */
27     private at.techmath.boltzmann.specification.ISpeciesSet _speciesSpecification;
28
29     /**
30      * Index der Spezies innerhalb der Spezifikation.
31      */
32     private int _index;
33
34     /**
35      * Gibt den Index der Spezies innerhalb der Spezifikation zurueck.
36      * @return Index der Spezies innerhalb der Spezifikation.
37      */
38     protected int getIndex() {
39         return _index;
40     } // protected int getIndex() ...
41
42     /**
43      * Sampler der Parameter-Spezies.
44      */
45     private ExponentialSpeciesSampler _parameterSampler = null;
46
47     /**
48      * Label, mit dem die Strukturen der Parameter-Spezies versehen sind.
49      */
50     private Label _parameterLabel = null;
51
52     /**
53      * Gibt eine Methode zum Sampeln der Spezies zurueck.
54      * @return Methode zum Sampeln der Spezies.
55      */
56     @Override
57     public SamplerMethod getMethod() {
58         if (_methodStack.empty()) {
59             return new SamplerMethodSet(this);
60         } else { // if (!_methodStack.empty()) ...
61             return _methodStack.pop();
62         } // if (!_methodStack.empty()) ...
63     } // public SamplerMethod getMethod() ...
64
65     /**
66      * Baut die Instanz zusammen. Es muessen alle referenzierten Sampler existieren.
67      * @throws IllegalArgumentException sampler ist null oder Spezifikation ist ungueltig.
68      */
69     @Override
70     protected void buildInstance(ExponentialSampler sampler) {
71         if (sampler == null) {
72             throw new IllegalArgumentException("sampler = null.");
73         } // if (sampler == null) ...
74         at.techmath.boltzmann.specification.ISpeciesReference speciesReference =
               _speciesSpecification.getSpeciesParameter();
75         if (speciesReference == null) {
```

```
76              throw new IllegalArgumentException("Spezies-Referenz der Parameter-Spezies der Set-
                    Spezies ist null.");
77          } // if (speciesReference == null) ...
78          // Spezies-Sampler setzen
79          at.techmath.boltzmann.specification.ISpecies species = speciesReference.getSpecies();
80          if (species == null) {
81              throw new IllegalArgumentException("Spezies-Referenz der Parameter-Spezies der Set-
                    Spezies referenziert keine Spezies.");
82          } // if (species == null) ...
83          if (species instanceof at.techmath.boltzmann.specification.ISpeciesOne) {
84              _parameterSampler = sampler.getSpeciesSamplerOne();
85          } else if (species instanceof at.techmath.boltzmann.specification.ISpeciesSingleton) {
86              _parameterSampler = sampler.getSpeciesSamplerSingleton();
87          } else if (species instanceof at.techmath.boltzmann.specification.INamedSpecies) {
88              _parameterSampler = sampler.getSamplerIndex(((at.techmath.boltzmann.specification.
                    INamedSpecies) species).getIndex());
89          } else {
90              throw new IllegalArgumentException("Unbekannter Spezies-Typ.");
91          }
92          // Label setzen
93          at.techmath.boltzmann.specification.ILabel label = speciesReference.getLabel();
94          if (label != null) {
95              _parameterLabel = sampler.getLabelIndex(label.getIndex());
96          } else { // if (label == null) ...
97              _parameterLabel = null;
98          } // if (label == null) ...
99      } // protected void buildInstance(ExponentialSampler sampler) ...
100
101     /**
102      * Gibt den Wert der erzeugenden Funktion an der Parameter-Stelle zurueck.
103      * @param context Sampling-Context.
104      * @return Wert der erzeugenden Funktion an der Parameter-Stelle.
105      * @throws IllegalArgumentException Wird geworfen, wenn context = null ist.
106      */
107     @Override
108     protected double getGeneratingFunctionValue(SamplingContext context) {
109         if (context == null) {
110             throw new IllegalArgumentException("context = null.");
111         } // if (context == null) ...
112         return context.getGeneratingFunctionValue(_index);
113     } // protected double getGeneratingFunctionValue(SamplingContext context) ...
114
115     /**
116      * Stack, der SamplerMethodSet-Instanzen zur weiteren Verwendung speichert.
117      */
118     private Stack<SamplerMethodSet> _methodStack = new Stack<SamplerMethodSet>();
119
120     /**
121      * Speichert die angegebene Methode zur Wiederverwendung.
122      * @param method Methode, die zur Wiederverwendung gespeichert werden soll.
123      * @throws IllegalArgumentException Wird geworfen, wenn method = null ist.
124      */
125     private void pushMethod(SamplerMethodSet method) {
126         if (method == null) {
127             throw new IllegalArgumentException("method = null.");
128         } // if (method == null) ...
129         method.reset();
130         _methodStack.push(method);
131     } // private void pushMethod(SamplerMethodSet method) ...
132
133     /**
134      * Sampler-Methode, die eine Struktur der Set-Spezies erzeugt.
135      * @author Stefan Schnabl (e0226245)
136      */
137     private class SamplerMethodSet extends SamplerMethod {
138         /**
139          * Erzeugt eine neue Instanz.
140          * @param parent Sampler, dem die Methode angehoert.
141          * @throws IllegalArgumentException parent = null.
142          */
143         public SamplerMethodSet(ExponentialSpeciesSamplerSet parent) {
144             super();
145             if (parent == null) {
146                 throw new IllegalArgumentException("parent = null.");
147             } // if (parent == null) ...
148             _parent = parent;
149         } // public SamplerMethodSet(ExponentialSpeciesSamplerSet parent) ...
150
151         /**
152          * Sampler, dem die Methode angehoert.
```

```
153          */
154         private ExponentialSpeciesSamplerSet _parent;
155 
156         /**
157          * Strukturen der Ergebnis-Struktur.
158          */
159         private Structure[] _resultStructures = null;
160 
161         /**
162          * Index der letzten bereits gesampelten Struktur.
163          * -1 bedeutet, dass noch keine Struktur gesampelt wurde.
164          */
165         private int _currentStructureIndex = -1;
166 
167         /**
168          * Setzt die Methode in den Anfangszustand zurueck.
169          */
170         @Override
171         public void reset() {
172             super.reset();
173             _resultStructures = null;
174             _currentStructureIndex = -1;
175         } // public void reset() ...
176 
177         /**
178          * Fuehrt die aktuelle Methode fort.
179          * @param context Sampling-Context.
180          * @throws IllegalArgumentException Der angegebene Context ist null.
181          * @throws IllegalStateException Die Methode ist in einem ungueltigen Zustand.
182          */
183         @Override
184         public void execute(SamplingContext context) {
185             if (context == null) {
186                 throw new IllegalArgumentException("context = null.");
187             } // if (context == null) ...
188             if (_currentStructureIndex < 0) {
189                 // Ermitteln der Groesse.
190                 int structureSize = context.generatePoissonRandomNumber(_parent.
                        _parameterSampler.getGeneratingFunctionValue(context));
191                 // Erzeugen des Ergebnis-Arrays.
192                 _resultStructures = new Structure[structureSize];
193                 if (structureSize <= 0) {
194                     Label label = getLabel();
195                     Structure resultStructure;
196                     if (label != null) {
197                         resultStructure = new LabeledStructureSet(_resultStructures);
198                         ((LabeledStructureSet) resultStructure).setLabel(label);
199                     } else { // if (label == null) ...
200                         resultStructure = new StructureSet(_resultStructures);
201                     } // if (label == null) ...
202                     context.setResult(resultStructure);
203                     context.removeMethod();
204                     _parent.pushMethod(this);
205                     return;
206                 } else { // if (structureSize > 0) ...
207                     _currentStructureIndex = 0;
208                     SamplerMethod firstSampler = _parent._parameterSampler.getMethod();
209                     firstSampler.setLabel(_parent._parameterLabel);
210                     context.pushMethod(firstSampler);
211                     return;
212                 } // if (structureSize > 0) ...
213             } else { // if (_currentStructureIndex >= 0) ...
214                 int structureSize = _resultStructures.length;
215                 if (_currentStructureIndex < structureSize - 1) {
216                     _resultStructures[_currentStructureIndex] = context.getResult();
217                     _currentStructureIndex++;
218                     SamplerMethod nextSampler = _parent._parameterSampler.getMethod();
219                     nextSampler.setLabel(_parent._parameterLabel);
220                     context.pushMethod(nextSampler);
221                     return;
222                 } else { // if (_currentStructureIndex >= structureSize - 1) ...
223                     _resultStructures[_currentStructureIndex] = context.getResult();
224                     Label label = getLabel();
225                     Structure resultStructure;
226                     if (label != null) {
227                         resultStructure = new LabeledStructureSet(_resultStructures);
228                         ((LabeledStructureSet) resultStructure).setLabel(label);
229                     } else { // if (label == null) ...
230                         resultStructure = new StructureSet(_resultStructures);
231                     } // if (label == null) ...
```

```
232                     context.setResult(resultStructure);
233                     context.removeMethod();
234                     _parent.pushMethod(this);
235                     return;
236                 } // if (_currentStructureIndex >= structureSize - 1) ...
237             } // if (_currentStructureIndex >= 0) ...
238         } // public void execute(SamplingContext context) ...
239     } // private class SamplerMethodSet extends SamplerMethod ...
240 } // public class ExponentialSpeciesSamplerSet extends ExponentialSpeciesSampler ...
```

Listing 79: Klasse *at.techmath.boltzmann.sampler.ExponentialSpeciesSamplerSingleton*

```
1  package at.techmath.boltzmann.sampler;
2
3  import java.util.Stack;
4
5  /**
6   * Sampler fuer Strukturen der Singleton-Spezies.
7   * @author Stefan Schnabl (e0226245)
8   */
9  class ExponentialSpeciesSamplerSingleton extends ExponentialSpeciesSampler {
10     /**
11      * Gibt eine Methode zum Sampeln der Spezies zurueck.
12      * @return Methode zum Sampeln der Spezies.
13      */
14     @Override
15     public SamplerMethod getMethod() {
16         if (_methodStack.empty()) {
17             return new SamplerMethodSingleton(this);
18         } else { // if (!_methodStack.empty()) ...
19             return _methodStack.pop();
20         } // if (!_methodStack.empty()) ...
21     } // public SamplerMethod getMethod() ...
22
23     /**
24      * Baut die Instanz zusammen. Es muessen alle referenzierten Sampler existieren.
25      * @throws IllegalArgumentException sampler ist null.
26      */
27     @Override
28     protected void buildInstance(ExponentialSampler sampler) {
29         if (sampler == null) {
30             throw new IllegalArgumentException("sampler = null.");
31         } // if (sampler == null) ...
32     } // protected void buildInstance(ExponentialSampler sampler) ...
33
34     /**
35      * Gibt den Wert der erzeugenden Funktion an der Parameter-Stelle zurueck.
36      * @param context Sampling-Context.
37      * @return Wert der erzeugenden Funktion an der Parameter-Stelle.
38      * @throws IllegalArgumentException Wird geworfen, wenn context = null ist.
39      */
40     @Override
41     protected double getGeneratingFunctionValue(SamplingContext context) {
42         if (context == null) {
43             throw new IllegalArgumentException("context = null.");
44         } // if (context == null) ...
45         return context.getBoltzmannParameter();
46     } // protected double getGeneratingFunctionValue(SamplingContext context) ...
47
48     /**
49      * Stack, der SamplerMethodSingleton-Instanzen zur weiteren Verwendung speichert.
50      */
51     private Stack<SamplerMethodSingleton> _methodStack = new Stack<SamplerMethodSingleton>();
52
53     /**
54      * Speichert die angegebene Methode zur Wiederverwendung.
55      * @param method Methode, die zur Wiederverwendung gespeichert werden soll.
56      * @throws IllegalArgumentException method = null.
57      */
58     private void pushMethod(SamplerMethodSingleton method) {
59         if (method == null) {
60             throw new IllegalArgumentException("method = null.");
61         } // if (method == null) ...
62         method.reset();
63         _methodStack.push(method);
64     } // private void pushMethod(SamplerMethodSingleton method) ...
65
66     /**
67      * Sampler-Methode, die eine Struktur der Singleton-Spezies erzeugt.
68      * @author Stefan Schnabl (e0226245)
69      */
70     private class SamplerMethodSingleton extends SamplerMethod {
71         /**
72          * Erzeugt eine neue Instanz.
73          * @param parent Sampler, dem die Methode angehoert.
74          * @throws IllegalArgumentException parent = null.
75          */
76         public SamplerMethodSingleton(ExponentialSpeciesSamplerSingleton parent) {
77             super();
78             if (parent == null) {
79                 throw new IllegalArgumentException("parent = null.");
```

```
80              } // if (parent == null) ...
81              _parent = parent;
82          } // public SamplerMethodSingleton(ExponentialSpeciesSamplerSingleton parent) ...
83
84          /**
85           * Sampler, dem die Methode angehoert.
86           */
87          private ExponentialSpeciesSamplerSingleton _parent;
88
89          /**
90           * Fuehrt die aktuelle Methode fort.
91           * @param context Sampling-Context.
92           * @throws IllegalArgumentException Der angegebene Context ist null.
93           * @throws IllegalStateException Die Methode ist in einem ungueltigen Zustand.
94           */
95          @Override
96          public void execute(SamplingContext context) {
97              if (context == null) {
98                  throw new IllegalArgumentException("context = null.");
99              } // if (context == null) ...
100             Label label = getLabel();
101             StructureSingleton structure;
102             if (label != null) {
103                 structure = new LabeledStructureSingleton();
104                 ((LabeledStructureSingleton) structure).setLabel(label);
105             } else { // if (label == null) ...
106                 structure = new StructureSingleton();
107             } // if (label == null) ...
108             context.setResult(structure);
109             context.incrementSize();
110             context.removeMethod();
111             _parent.pushMethod(this);
112         } // public void execute(SamplingContext context) ...
113     } // private class SamplerMethodSingleton extends SamplerMethod ...
114 } // class ExponentialSpeciesSamplerSingleton extends ExponentialSpeciesSampler ...
```

Listing 80: Klasse *at.techmath.boltzmann.sampler.ExponentialSpeciesSamplerSum*

```
1  package at.techmath.boltzmann.sampler;
2
3  import java.util.Stack;
4
5  /**
6   * Sampler fuer Strukturen der Summen-Spezies.
7   * @author Stefan Schnabl (e0226245)
8   */
9  class ExponentialSpeciesSamplerSum extends ExponentialSpeciesSampler {
10     /**
11      * Erzeugt eine neue Instanz.
12      * @param speciesSpecification Spezifikation der Spezies.
13      * @throws IllegalArgumentException speciesSpecification ist null.
14      */
15     public ExponentialSpeciesSamplerSum(at.techmath.boltzmann.specification.ISpeciesSum
            speciesSpecification) {
16         super();
17         if (speciesSpecification == null) {
18             throw new IllegalArgumentException("speciesSpecification = null.");
19         } // if (speciesSpecification == null) ...
20         _speciesSpecification = speciesSpecification;
21         _index = speciesSpecification.getIndex();
22     } // public ExponentialSpeciesSamplerSum(at.techmath.boltzmann.specification.ISpeciesSum
            speciesSpecification) ...
23
24     /**
25      * Spezifikation der Spezies.
26      */
27     private at.techmath.boltzmann.specification.ISpeciesSum _speciesSpecification;
28
29     /**
30      * Index der Spezies innerhalb der Spezifikation.
31      */
32     private int _index;
33
34     /**
35      * Gibt den Index der Spezies innerhalb der Spezifikation zurueck.
36      * @return Index der Spezies innerhalb der Spezifikation.
37      */
38     protected int getIndex() {
39         return _index;
40     } // protected int getIndex() ...
41
42     /**
43      * Liste der Sampler, aus denen sich die Summe zusammensetzt.
44      */
45     private ExponentialSpeciesSampler[] _samplers = null;
46
47     /**
48      * Liste der Labels, mit denen die Summanden versehen werden.
49      */
50     private Label[] _labels = null;
51
52     /**
53      * Gibt eine Methode zum Sampeln der Spezies zurueck.
54      * @return Methode zum Sampeln der Spezies.
55      */
56     @Override
57     public SamplerMethod getMethod() {
58         if (_methodStack.empty()) {
59             return new SamplerMethodSum(this);
60         } else { // if (!_methodStack.empty()) ...
61             return _methodStack.pop();
62         } // if (!_methodStack.empty()) ...
63     } // public SamplerMethod getMethod() ...
64
65     /**
66      * Baut die Instanz zusammen. Es muessen alle referenzierten Sampler existieren.
67      * @throws IllegalArgumentException sampler ist null oder Spezifikation ist ungueltig.
68      */
69     @Override
70     protected void buildInstance(ExponentialSampler sampler) {
71         if (sampler == null) {
72             throw new IllegalArgumentException("sampler = null.");
73         } // if (sampler == null) ...
74         int count = _speciesSpecification.getAddendCount();
75         _samplers = new ExponentialSpeciesSampler[count];
76         _labels = new Label[count];
77         for (int c = 0; c < count; c++) {
```

```
78              at.techmath.boltzmann.specification.ISpeciesReference speciesReference =
                    _speciesSpecification.getAddendIndex(c);
79              if (speciesReference == null) {
80                  throw new IllegalArgumentException("Spezies-Referenz eines Summanden der Summen
                        -Spezies ist null.");
81              } // if (speciesReference == null) ...
82              // Spezies-Sampler setzen.
83              at.techmath.boltzmann.specification.ISpecies species = speciesReference.getSpecies
                    ();
84              if (species == null) {
85                  throw new IllegalArgumentException("Spezies-Referenz eines Summanden der Summen
                        -Spezies referenziert keine Spezies.");
86              } // if (species == null) ...
87              if (species instanceof at.techmath.boltzmann.specification.ISpeciesOne) {
88                  _samplers[c] = sampler.getSpeciesSamplerOne();
89              } else if (species instanceof at.techmath.boltzmann.specification.ISpeciesSingleton
                    ) {
90                  _samplers[c] = sampler.getSpeciesSamplerSingleton();
91              } else if (species instanceof at.techmath.boltzmann.specification.INamedSpecies) {
92                  _samplers[c] = sampler.getSamplerIndex(((at.techmath.boltzmann.specification.
                        INamedSpecies) species).getIndex());
93              } else {
94                  throw new IllegalArgumentException("Unbekannter Spezies-Typ.");
95              }
96              // Label setzen
97              at.techmath.boltzmann.specification.ILabel label = speciesReference.getLabel();
98              if (label != null) {
99                  _labels[c] = sampler.getLabelIndex(label.getIndex());
100             } else { // if (label == null) ...
101                 _labels[c] = null;
102             } // if (label == null) ...
103         } // for (int c = 0; c < count; c++) ...
104     } // protected void buildInstance(ExponentialSampler sampler) ...
105
106     /**
107      * Gibt den Wert der erzeugenden Funktion an der Parameter-Stelle zurueck.
108      * @param context Sampling-Context.
109      * @return Wert der erzeugenden Funktion an der Parameter-Stelle.
110      * @throws IllegalArgumentException Wird geworfen, wenn context = null ist.
111      */
112     @Override
113     protected double getGeneratingFunctionValue(SamplingContext context) {
114         if (context == null) {
115             throw new IllegalArgumentException("context = null.");
116         } // if (context == null) ...
117         return context.getGeneratingFunctionValue(_index);
118     } // protected double getGeneratingFunctionValue(SamplingContext context) ...
119
120     /**
121      * Stack, der SamplerMethodSum-Instanzen zur weiteren Verwendung speichert.
122      */
123     private Stack<SamplerMethodSum> _methodStack = new Stack<SamplerMethodSum>();
124
125     /**
126      * Speichert die angegebene Methode zur Wiederverwendung.
127      * @param method Methode, die zur Wiederverwendung gespeichert werden soll.
128      * @throws IllegalArgumentException method = null.
129      */
130     private void pushMethod(SamplerMethodSum method) {
131         if (method == null) {
132             throw new IllegalArgumentException("method = null.");
133         } // if (method == null) ...
134         method.reset();
135         _methodStack.push(method);
136     } // private void pushMethod(SamplerMethodSum method) ...
137
138     /**
139      * Sampler-Methode, die eine Struktur der Summen-Spezies erzeugt.
140      * @author Stefan Schnabl (e0226245)
141      */
142     private class SamplerMethodSum extends SamplerMethod {
143         /**
144          * Erzeugt eine neue Instanz.
145          * @param parent Sampler, dem die Methode angehoert.
146          * @throws IllegalArgumentException parent = null.
147          */
148         public SamplerMethodSum(ExponentialSpeciesSamplerSum parent) {
149             super();
150             if (parent == null) {
151                 throw new IllegalArgumentException("parent = null.");
```

```
152             } // if (parent == null) ...
153             _parent = parent;
154         } // public SamplerMethodSum(ExponentialSpeciesSamplerSum parent) ...
155 
156         /**
157          * Sampler, dem die Methode angehoert.
158          */
159         private ExponentialSpeciesSamplerSum _parent;
160 
161         /**
162          * Initialisierungs-Zustand, beim Aufruf wird der Summand ermittelt
163          * und der Sampler dieses Summanden aufgerufen. Die Methode wird
164          * in den Zustand STATE_RETURN_SPECIES versetzt.
165          */
166         private static final int STATE_INIT = 0;
167         /*
168          * Zustand, in dem der gesampelte Summand zurueckgegeben wird.
169          * Die Methode ist dann fertig, entfernt sich vom Stack und
170          * geht in den Zustand STATE_FINISHED ueber.
171          */
172         private static final int STATE_RETURN_SPECIES = 1;
173 
174         /**
175          * Aktueller Zustand der Methode.
176          */
177         private int _currentState = STATE_INIT;
178 
179         /**
180          * Setzt die Methode in den Anfangszustand zurueck.
181          */
182         @Override
183         public void reset() {
184             super.reset();
185             _currentState = STATE_INIT;
186         } // public void reset() ...
187 
188         /**
189          * Fuehrt die Methode im Zustand STATE_INIT aus.
190          * @param context Sampling-Context.
191          */
192         private void doStateInit(SamplingContext context) {
193             // Die Spezies i wird mit Wahrscheinlichkeit A_i(x) / Summe A_i(x) gezogen. (i = 0
                    .. addendCount - 1)
194             // Inversenmethode:
195             // Waehle eine Zufallszahl p auf [0, 1) gleichverteilt und ermittle
196             // das Minimum { k in Nat0 : Summe (i = 0 .. k) A_i(x) / Summe A_i(x) >= p } =
197             // Minimum { k in Nat0 : Summe (i = 0 .. k) A_i(x) >= p * Summe A_i(x) },
198             // und sample die i-te Spezies.
199             double randomNumber = context.generateUniformRandomNumber() * context.
                        getGeneratingFunctionValue(_index);
200             int addendCount = _parent._samplers.length;
201             int currentAddendIndex = -1;
202             ExponentialSpeciesSampler currentSampler = null;
203             double sum = 0.0D;
204             do {
205                 currentAddendIndex++;
206                 currentSampler = _parent._samplers[currentAddendIndex];
207                 sum += currentSampler.getGeneratingFunctionValue(context);
208             } while (sum < randomNumber && currentAddendIndex < addendCount - 1);
209 
210             // Sampeln der currentAddendIndex-Spezies
211             SamplerMethod samplerMethod = currentSampler.getMethod();
212             samplerMethod.setLabel(_parent._labels[currentAddendIndex]);
213             context.pushMethod(samplerMethod);
214             _currentState = STATE_RETURN_SPECIES;
215         } // private void doStateInit() ...
216 
217         /**
218          * Fuehrt die Methode im Zustand STATE_RETURN_SPECIES aus.
219          * @param context Sampling-Context.
220          */
221         private void doStateReturnSpecies(SamplingContext context) {
222             Structure sampledStructure = context.getResult();
223             Structure resultStructure;
224             Label label = getLabel();
225             if (label != null) {
226                 LabeledStructureSum structureSum = new LabeledStructureSum(sampledStructure);
227                 structureSum.setLabel(label);
228                 resultStructure = structureSum;
229             } else { // if (label == null) ...
```

```
230                 resultStructure = sampledStructure;
231             } // if (label == null) ...
232             context.setResult(resultStructure);
233             context.removeMethod();
234             _parent.pushMethod(this);
235         } // private void doStateReturnSpecies() ...
236 
237         /**
238          * Fuehrt die aktuelle Methode fort.
239          * @param context Sampling-Context.
240          * @throws IllegalArgumentException Der angegebene Context ist null.
241          * @throws IllegalStateException Die Methode ist in einem ungueltigen Zustand.
242          */
243         @Override
244         public void execute(SamplingContext context) {
245             if (context == null) {
246                 throw new IllegalArgumentException("context = null.");
247             } // if (context == null) ...
248 
249             switch (_currentState) {
250                 case STATE_INIT:
251                     doStateInit(context);
252                     break;
253                 case STATE_RETURN_SPECIES:
254                     doStateReturnSpecies(context);
255                     break;
256                 default: throw new IllegalStateException("SamplerMethodSum in ungueltigem
                        Zustand.");
257             } // switch (_currentState) ...
258         } // public void execute(SamplingContext context) ...
259     } // private class SamplerMethodSum extends SamplerMethod ...
260 } // class ExponentialSpeciesSamplerSum extends ExponentialSpeciesSampler ...
```

Listing 81: Interface *at.techmath.boltzmann.sampler.IExponentialSampler*

```
1  package at.techmath.boltzmann.sampler;
2
3  /**
4   * Repraesentiert einen exponentiellen Boltzmann-Sampler.
5   * @author Stefan Schnabl (e0226245)
6   */
7  public interface IExponentialSampler {
8      /**
9       * Sampelt eine Struktur.
10      * @param x Parameter des Samplers.
11      * @return Struktur, die gesampelt wurde.
12      * @throws at.techmath.boltzmann.sampler.StructureSizeExceededException Wird geworfen,
13      *   wenn die maximal zulaessige Strukturgroesse ueberschritten wurde.
14      */
15     ISamplingResult sample(double x) throws StructureSizeExceededException;
16
17     /**
18      * Setzt die maximal zulaessige Groesse der gesampelten Strukturen.
19      * Der Standardwert nach Erstellung der Instanz ist Long.MAX_VALUE.
20      * @param maxSize Maximal zulaessige Groesse der gesampelten Strukturen.
21      * @throws IllegalArgumentException maxSize ist kleiner als 0.
22      */
23     void setMaxSize(long maxSize);
24
25     /**
26      * Gibt die maximal zulaessige Groesse der gesampelten Strukturen zurueck.
27      * @return Maximal zulaessige Groesse der gesampelten Strukturen.
28      */
29     long getMaxSize();
30 } // public interface IExponentialSampler ...
```

Listing 82: Interface *at.techmath.boltzmann.sampler.IExponentialSamplerFactory*

```
1  package at.techmath.boltzmann.sampler;
2
3  import at.techmath.boltzmann.specification.INamedSpecies;
4
5  /**
6   * Factory, die aus kombinatorischen Spezifikationen
7   * exponentielle Boltzmann-Sampler erzeugen kann.
8   * @author Stefan Schnabl (e0226245)
9   */
10 public interface IExponentialSamplerFactory {
11     /**
12      * Erzeugt fuer die angegebenen Spezies
13      * einen exponentiellen Boltzmann-Sampler.
14      * @param species Spezies, fuer die der Sampler erzeugt werden soll.
15      * @param atomFactory Factory, die die Grundmenge erzeugt. Falls dieser Parameter
16      *   null ist, dann erzeugt der generierte Sampler Atome aus der Menge der natuerlichen
            Zahlen.
17      * @return Exponentieller Boltzmann-Sampler fuer die angegebene Spezies.
18      * @throws IllegalArgumentException Wird geworfen, wenn die
19      *   uebergebene Spezies null ist.
20      */
21     IExponentialSampler createSampler(INamedSpecies species, IAtomFactory atomFactory);
22 } // public interface IExponentialSamplerFactory ...
```

Listing 83: Interface *at.techmath.boltzmann.sampler.IIntegerLabel*

```
1  package at.techmath.boltzmann.sampler;
2
3  /**
4   * Repraesentiert ein Zahlen-Label.
5   * @author Stefan Schnabl (e0226245)
6   */
7  public interface IIntegerLabel extends ILabel {
8      /**
9       * Gibt die Nummer des Zahlen-Labels zurueck.
10      * @return Nummer des Zahlen-Labels.
11      */
12     long getNumber();
13 } // public interface IIntegerLabel extends ILabel ...
```

Listing 84: Interface *at.techmath.boltzmann.sampler.ILabel*

```
1 package at.techmath.boltzmann.sampler;
2
3 /**
4  * Label, mit dem eine Struktur versehen werden kann.
5  * @author Stefan Schnabl (e0226245)
6  */
7 public interface ILabel {
8 } // public interface ILabel ...
```

Listing 85: Interface *at.techmath.boltzmann.sampler.ILabeledStructure*

```
1  package at.techmath.boltzmann.sampler;
2
3  /**
4   * Repraesentiert eine Struktur, die mit einem Label versehen ist.
5   * @author Stefan Schnabl (e0226245)
6   */
7  public interface ILabeledStructure extends IStructure {
8      /**
9       * Gibt das Label zurueck, mit dem die Struktur versehen ist.
10      * @return Label, mit dem die Struktur versehen ist.
11      */
12     ILabel getLabel();
13 } // public interface ILabeledStructure extends IStructure ...
```

Listing 86: Klasse *at.techmath.boltzmann.sampler.IntegerLabel*

```
1  package at.techmath.boltzmann.sampler;
2
3  /**
4   * Numerisches Label.
5   * @author Stefan Schnabl (e0226245)
6   */
7  class IntegerLabel extends Label implements IIntegerLabel {
8      /**
9       * Erzeugt eine neue Instanz.
10      * @param number Nummer des Zahlen-Labels.
11      */
12     public IntegerLabel(long number) {
13         super();
14         _number = number;
15     } // public IntegerLabel(long number) ...
16
17     /**
18      * Nummer des Zahlen-Labels.
19      */
20     private long _number;
21
22     /**
23      * Gibt die Nummer des Zahlen-Labels zurueck.
24      * @return Nummer des Zahlen-Labels.
25      */
26     public long getNumber() {
27         return _number;
28     } // public long getNumber() ...
29 } // class IntegerLabel extends Label implements IIntegerLabel ...
```

Listing 87: Klasse *at.techmath.boltzmann.sampler.IntegerLabelingContext*

```
1  package at.techmath.boltzmann.sampler;
2
3  import java.util.Stack;
4  import at.techmath.boltzmann.random.IUniformRandomGenerator;
5
6  /**
7   * Labeling-Context fuer die Verteilung der natuerlichen Zahlen
8   * der Grundmenge auf die Strukturen.
9   * @author Stefan Schnabl (e0226245)
10  */
11 class IntegerLabelingContext {
12     /**
13      * Erzeugt eine neue Instanz.
14      * @param size Groesse der Struktur, die gelabelt werden soll.
15      * @param uniformRandomGenerator Zufallszahlengenerator.
16      * @throws IllegalArgumentException size nicht im gueltigen Bereich oder
            uniformRandomGenerator ist null.
17      */
18     public IntegerLabelingContext(long size, IUniformRandomGenerator uniformRandomGenerator) {
19         super();
20         if (size >= Integer.MAX_VALUE || size < 0L) {
21             throw new IllegalArgumentException("size nicht im gueltigen Bereich.");
22         } // if (size >= Integer.MAX_VALUE || size < 0L) ...
23         if (uniformRandomGenerator == null) {
24             throw new IllegalArgumentException("uniformRandomGenerator = null.");
25         } // if (uniformRandomGenerator == null) ...
26         _uniformRandomGenerator = uniformRandomGenerator;
27         int intSize = (int) size;
28         _atomArray = new int[intSize];
29         _remainingAtomCount = intSize;
30         for (int c = 0; c < intSize; c++) {
31             _atomArray[c] = c + 1;
32         } // for (int c = 0; c < intSize; c++) ...
33     } // public IntegerLabelingContext(long size) ...
34
35     /**
36      * Stack, der IntegerLabelingMethodSum-Instanzen zur Wiederverwendung speichert.
37      */
38     private Stack<IntegerLabelingMethodSum> _methodSumStack = new Stack<
           IntegerLabelingMethodSum>();
39
40     /**
41      * Gibt eine zur Verwendung bereite IntegerLabelingMethodSum-Instanz zurueck.
42      * @param structure Struktur, mit der die Methode aufgerufen wird.
43      * @return IntegerLabelingMethodSum-Instanz.
44      * @throws IllegalArgumentException structure ist null.
45      */
46     public IntegerLabelingMethodSum getMethodSum(StructureSum structure) {
47         if (structure == null) {
48             throw new IllegalArgumentException("structure = null.");
49         } // if (structure == null) ...
50         IntegerLabelingMethodSum result =  _methodSumStack.empty() ?
51             new IntegerLabelingMethodSum() : _methodSumStack.pop();
52         result.setStructure(structure);
53         return result;
54     } // public IntegerLabelingMethodSum getMethodSum() ...
55
56     /**
57      * Speichert die angegebene Methode zur weiteren Verwendung.
58      * @param method Methode, die gespeichert werden soll.
59      * @throws IllegalArgumentException method ist null.
60      */
61     public void releaseMethodSum(IntegerLabelingMethodSum method) {
62         if (method == null) {
63             throw new IllegalArgumentException("method = null.");
64         } // if (method == null) ...
65         method.reset();
66         _methodSumStack.push(method);
67     } // public void releaseMethodSum(IntegerLabelingMethodSum method) ...
68
69     /**
70      * Stack, der IntegerLabelingMethodProduct-Instanzen zur Wiederverwendung speichert.
71      */
72     private Stack<IntegerLabelingMethodProduct> _methodProductStack = new Stack<
           IntegerLabelingMethodProduct>();
73
74     /**
75      * Gibt eine zur Verwendung bereite IntegerLabelingMethodProduct-Instanz zurueck.
76      * @param structure Struktur, mit der die Methode aufgerufen wird.
```

```
77       * @return IntegerLabelingMethodProduct-Instanz.
78       * @throws IllegalArgumentException structure ist null.
79       */
80      public IntegerLabelingMethodProduct getMethodProduct(StructureProduct structure) {
81          if (structure == null) {
82              throw new IllegalArgumentException("structure = null.");
83          } // if (structure == null) ...
84          IntegerLabelingMethodProduct result =  _methodProductStack.empty() ?
85              new IntegerLabelingMethodProduct() : _methodProductStack.pop();
86          result.setStructure(structure);
87          return result;
88      } // public IntegerLabelingMethodProduct getMethodProduct() ...
89
90      /**
91       * Speichert die angegebene Methode zur weiteren Verwendung.
92       * @param method Methode, die gespeichert werden soll.
93       * @throws IllegalArgumentException method ist null.
94       */
95      public void releaseMethodProduct(IntegerLabelingMethodProduct method) {
96          if (method == null) {
97              throw new IllegalArgumentException("method = null.");
98          } // if (method == null) ...
99          method.reset();
100         _methodProductStack.push(method);
101     } // public void releaseMethodProduct(IntegerLabelingMethodProduct method) ...
102
103     /**
104      * Stack, der IntegerLabelingMethodSequence-Instanzen zur Wiederverwendung speichert.
105      */
106     private Stack<IntegerLabelingMethodSequence> _methodSequenceStack = new Stack<
          IntegerLabelingMethodSequence>();
107
108     /**
109      * Gibt eine zur Verwendung bereite IntegerLabelingMethodSequence-Instanz zurueck.
110      * @param structure Struktur, mit der die Methode aufgerufen wird.
111      * @return IntegerLabelingMethodSequence-Instanz.
112      * @throws IllegalArgumentException structure ist null.
113      */
114     public IntegerLabelingMethodSequence getMethodSequence(StructureSequence structure) {
115         if (structure == null) {
116             throw new IllegalArgumentException("structure = null.");
117         } // if (structure == null) ...
118         IntegerLabelingMethodSequence result = _methodSequenceStack.empty() ?
119             new IntegerLabelingMethodSequence() : _methodSequenceStack.pop();
120         result.setStructure(structure);
121         return result;
122     } // public IntegerLabelingMethodSequence getMethodSequence(StructureSequence structure)
          ...
123
124     /**
125      * Speichert die angegebene Methode zur weiteren Verwendung.
126      * @param method Methode, die gespeichert werden soll.
127      * @throws IllegalArgumentException method ist null.
128      */
129     public void releaseMethodSequence(IntegerLabelingMethodSequence method) {
130         if (method == null) {
131             throw new IllegalArgumentException("method = null.");
132         } // if (method == null) ...
133         method.reset();
134         _methodSequenceStack.push(method);
135     } // public void releaseMethodSequence(IntegerLabelingMethodSequence method) ...
136
137     /**
138      * Stack, der IntegerLabelingMethodSet-Instanzen zur Wiederverwendung speichert.
139      */
140     private Stack<IntegerLabelingMethodSet> _methodSetStack = new Stack<
          IntegerLabelingMethodSet>();
141
142     /**
143      * Gibt eine zur Verwendung bereite IntegerLabelingMethodSet-Instanz zurueck.
144      * @param structure Struktur, mit der die Methode aufgerufen wird.
145      * @return IntegerLabelingMethodSet-Instanz.
146      * @throws IllegalArgumentException structure ist null.
147      */
148     public IntegerLabelingMethodSet getMethodSet(StructureSet structure) {
149         if (structure == null) {
150             throw new IllegalArgumentException("structure = null.");
151         } // if (structure == null) ...
152         IntegerLabelingMethodSet result = _methodSetStack.empty() ?
153             new IntegerLabelingMethodSet() : _methodSetStack.pop();
```

```
154         result.setStructure(structure);
155         return result;
156     } // public IntegerLabelingMethodSet getMethodSet(StructureSet structure) ...
157 
158     /**
159      * Speichert die angegebene Methode zur weiteren Verwendung.
160      * @param method Methode, die gespeichert werden soll.
161      * @throws IllegalArgumentException method ist null.
162      */
163     public void releaseMethodSet(IntegerLabelingMethodSet method) {
164         if (method == null) {
165             throw new IllegalArgumentException("method = null.");
166         } // if (method == null) ...
167         method.reset();
168         _methodSetStack.push(method);
169     } // public void releaseMethodSet(IntegerLabelingMethodSet method) ...
170 
171     /**
172      * Stack, der IntegerLabelingMethodCycle-Instanzen zur Wiederverwendung speichert.
173      */
174     private Stack<IntegerLabelingMethodCycle> _methodCycleStack = new Stack<
          IntegerLabelingMethodCycle>();
175 
176     /**
177      * Gibt eine zur Verwendung bereite IntegerLabelingMethodCycle-Instanz zurueck.
178      * @param structure Struktur, mit der die Methode aufgerufen wird.
179      * @return IntegerLabelingMethodCycle-Instanz.
180      * @throws IllegalArgumentException structure ist null.
181      */
182     public IntegerLabelingMethodCycle getMethodCycle(StructureCycle structure) {
183         if (structure == null) {
184             throw new IllegalArgumentException("structure = null.");
185         } // if (structure == null) ...
186         IntegerLabelingMethodCycle result = _methodCycleStack.empty() ?
187             new IntegerLabelingMethodCycle() : _methodCycleStack.pop();
188         result.setStructure(structure);
189         return result;
190     } // public IntegerLabelingMethodCycle getMethodCycle(StructureCycle structure) ...
191 
192     /**
193      * Speichert die angegebene Methode zur weiteren Verwendung.
194      * @param method Methode, die gespeichert werden soll.
195      * @throws IllegalArgumentException method ist null.
196      */
197     public void releaseMethodCycle(IntegerLabelingMethodCycle method) {
198         if (method == null) {
199             throw new IllegalArgumentException("method = null.");
200         } // if (method == null) ...
201         method.reset();
202         _methodCycleStack.push(method);
203     } // public void releaseMethodCycle(IntegerLabelingMethodCycle method) ...
204 
205     /**
206      * Stack, der IntegerLabelingMethodOne-Instanzen zur Wiederverwendung speichert.
207      */
208     private Stack<IntegerLabelingMethodOne> _methodOneStack = new Stack<
          IntegerLabelingMethodOne>();
209 
210     /**
211      * Gibt eine zur Verwendung bereite IntegerLabelingMethodOne-Instanz zurueck.
212      * @param structure Struktur, mit der die Methode aufgerufen wird.
213      * @return IntegerLabelingMethodOne-Instanz.
214      * @throws IllegalArgumentException structure ist null.
215      */
216     public IntegerLabelingMethodOne getMethodOne(StructureOne structure) {
217         if (structure == null) {
218             throw new IllegalArgumentException("structure = null.");
219         } // if (structure == null) ...
220         IntegerLabelingMethodOne result = _methodOneStack.empty() ?
221             new IntegerLabelingMethodOne() : _methodOneStack.pop();
222         return result;
223     } // public IntegerLabelingMethodOne getMethodOne(StructureOne structure) ...
224 
225     /**
226      * Speichert die angegebene Methode zur weiteren Verwendung.
227      * @param method Methode, die gespeichert werden soll.
228      * @throws IllegalArgumentException method ist null.
229      */
230     public void releaseMethodOne(IntegerLabelingMethodOne method) {
231         if (method == null) {
```

```
232             throw new IllegalArgumentException("method = null.");
233         } // if (method == null) ...
234         method.reset();
235         _methodOneStack.push(method);
236     } // public void releaseMethodOne(IntegerLabelingMethodOne method) ...
237
238     /**
239      * Stack, der IntegerLabelingMethodSingleton-Instanzen zur Wiederverwendung speichert.
240      */
241     private Stack<IntegerLabelingMethodSingleton> _methodSingletonStack = new Stack<
            IntegerLabelingMethodSingleton>();
242
243     /**
244      * Gibt eine zur Verwendung bereite IntegerLabelingMethodSingleton-Instanz zurueck.
245      * @param structure Struktur, mit der die Methode aufgerufen wird.
246      * @return IntegerLabelingMethodSingleton-Instanz.
247      * @throws IllegalArgumentException structure ist null.
248      */
249     public IntegerLabelingMethodSingleton getMethodSingleton(StructureIntegerSingleton
            structure) {
250         if (structure == null) {
251             throw new IllegalArgumentException("structure = null.");
252         } // if (structure == null) ...
253         IntegerLabelingMethodSingleton result = _methodSingletonStack.empty() ?
254             new IntegerLabelingMethodSingleton() : _methodSingletonStack.pop();
255         result.setStructure(structure);
256         return result;
257     } // public IntegerLabelingMethodSingleton getMethodSingleton(StructureIntegerSingleton
            structure) ...
258
259     /**
260      * Speichert die angegebene Methode zur weiteren Verwendung.
261      * @param method Methode, die gespeichert werden soll.
262      * @throws IllegalArgumentException method ist null.
263      */
264     public void releaseMethodSingleton(IntegerLabelingMethodSingleton method) {
265         if (method == null) {
266             throw new IllegalArgumentException("method = null.");
267         } // if (method == null) ...
268         method.reset();
269         _methodSingletonStack.push(method);
270     } // public void releaseMethodSingleton(IntegerLabelingMethodSingleton method) ...
271
272     /**
273      * Gibt zur angegebenen Struktur die entsprechende Labeling-Methode zurueck.
274      * @param structure Struktur, zu der die Labeling-Methode zurueckgegeben werden soll.
275      * @return Labeling-Methode zur angegebenen Struktur.
276      * @throws IllegalArgumentException Angegebene Struktur ist null oder von
277      *   unbekanntem Typ.
278      */
279     public IntegerLabelingMethod getLabelingMethod(Structure structure) {
280         if (structure == null) {
281             throw new IllegalArgumentException("structure = null.");
282         } // if (structure == null) ...
283         if (structure instanceof StructureIntegerSingleton) {
284             return getMethodSingleton((StructureIntegerSingleton) structure);
285         } else if (structure instanceof StructureOne) {
286             return getMethodOne((StructureOne) structure);
287         } else if (structure instanceof StructureSum) {
288             return getMethodSum((StructureSum) structure);
289         } else if (structure instanceof StructureProduct) {
290             return getMethodProduct((StructureProduct) structure);
291         } else if (structure instanceof StructureSequence) {
292             return getMethodSequence((StructureSequence) structure);
293         } else if (structure instanceof StructureSet) {
294             return getMethodSet((StructureSet) structure);
295         } else if (structure instanceof StructureCycle) {
296             return getMethodCycle((StructureCycle) structure);
297         } else {
298             throw new IllegalArgumentException("Struktur von unbekanntem Typ.");
299         }
300     } // public IntegerLabelingMethod getLabelingMethod(Structure structure) ...
301
302     /**
303      * Stack der Labeling-Methoden, die gerade ausgefuehrt werden.
304      */
305     private Stack<IntegerLabelingMethod> _labelingMethodStack = new Stack<IntegerLabelingMethod
            >();
306
307     /**
```

```
308      * Gibt zurueck, ob der Labeling-Methoden-Stack leer ist.
309      * @return Ist der Labeling-Methoden-Stack leer?
310      */
311     public boolean isEmpty() {
312         return _labelingMethodStack.empty();
313     } // public boolean isEmpty() ...
314 
315     /**
316      * Entfernt die zuletzt ausgefuehrte Methode.
317      */
318     public void removeMethod() {
319         _labelingMethodStack.pop();
320     } // public void removeMethod() ...
321 
322     /**
323      * Legt die angegebene Methode auf den Methoden-Stack.
324      * @param method Methode, die auf den Methoden-Stack gelegt werden soll.
325      * @throws IllegalArgumentException Die angegebene Methode ist null.
326      */
327     public void pushMethod(IntegerLabelingMethod method) {
328         if (method == null) throw new IllegalArgumentException("method = null.");
329         _labelingMethodStack.push(method);
330     } // public void pushMethod(IntegerLabelingMethod method) ...
331 
332     /**
333      * Gibt die aktuelle Methode des Methoden-Stacks zurueck.
334      * @return Aktuelle Methode des Methoden-Stacks.
335      */
336     public IntegerLabelingMethod peekMethod() {
337         return _labelingMethodStack.peek();
338     } // public SamplerMethod peekMethod() ...
339 
340     /**
341      * Zufallszahlengenerator.
342      */
343     private IUniformRandomGenerator _uniformRandomGenerator;
344 
345     /**
346      * Array der zu verteilenden Atome.
347      */
348     private int[] _atomArray;
349 
350     /**
351      * Anzahl der noch verfuegbaren Atome.
352      */
353     private int _remainingAtomCount;
354 
355     /**
356      * Gibt das nachste Atom zurueck.
357      * @return Naechstes Atom.
358      * @throws IllegalStateException Es stehen keine Atome mehr zur Verfuegung.
359      */
360     public long getNextAtom() {
361         int remainingSize = _remainingAtomCount;
362         if (remainingSize <= 0) {
363             throw new IllegalStateException("Es stehen keine Atome mehr zur Verfuegung.");
364         } else { // if (_remainingAtomCount > 0) ...
365             // Einen gleichverteilten Index ins noch zur Verfuegung stehende Array erzeugen.
366             // Damit die Gleichverteilung zustande kommt, werden Zufallszahlen aus dem
367             // Rest des Wertebereichs am Ende des Long-Ranges verworfen.
368             int nextIndex = -1;
369             while (nextIndex < 0) {
370                 long random = _uniformRandomGenerator.getNonNegativeLong();
371                 if (random < (Long.MAX_VALUE / remainingSize) * remainingSize) {
372                     nextIndex = (int) (random % remainingSize);
373                 } // if (random < (Long.MAX_VALUE / remainingSize) * remainingSize) ...
374             } // while (nextIndex < 0) ...
375             // Der Wert an diesem Index ist der Rueckgabewert.
376             // Er wird durch den am Ende des Feldes stehenden Wert ersetzt
377             // und das Feld wird um 1 Element verkleinert.
378             int result = _atomArray[nextIndex];
379             _atomArray[nextIndex] = _atomArray[remainingSize - 1];
380             _atomArray[remainingSize - 1] = 0;
381             _remainingAtomCount--;
382             return result;
383         } // if (_remainingAtomCount > 0) ...
384     } // public long getNextAtom() ...
385 } // class IntegerLabelingContext ...
```

Listing 88: Klasse *at.techmath.boltzmann.sampler.IntegerLabelingMethod*

```
1  package at.techmath.boltzmann.sampler;
2
3  /**
4  * Basisklasse fuer Methoden, die die Atome der Grundmenge
5  * der Struktur verteilen.
6  * @author Stefan Schnabl (e0226245)
7  */
8  abstract class IntegerLabelingMethod {
9      /**
10      * Setzt die Methode in den Anfangszustand zurueck.
11      */
12     public void reset() {
13     } // public void reset() ...
14
15     /**
16      * Fuehrt die aktuelle Methode fort.
17      * @param context Labeling-Context.
18      * @throws IllegalArgumentException Der angegebene Context ist null.
19      * @throws IllegalStateException Die Methode in einem ungueltigen Zustand.
20      */
21     public abstract void execute(IntegerLabelingContext context);
22 } // abstract class IntegerLabelingMethode ...
```

Listing 89: Klasse *at.techmath.boltzmann.sampler.IntegerLabelingMethodCycle*

```
1  package at.techmath.boltzmann.sampler;
2
3  /**
4   * Methode, die Integer-Label auf Cycle-Strukturen verteilt.
5   * @author Stefan Schnabl (e0226245)
6   */
7  class IntegerLabelingMethodCycle extends IntegerLabelingMethod {
8      /**
9       * Cycle-Struktur, die gerade gelabelt wird.
10      */
11     private StructureCycle _structure = null;
12
13     /**
14      * Setzt die Struktur, die gerade gelabelt wird.
15      * @param structure Struktur, die gerade gelabelt wird.
16      */
17     public void setStructure(StructureCycle structure) {
18         _structure = structure;
19     } // public void setStructure(StructureCycle structure) ...
20
21     /**
22      * Index des zuletzt gelabelten Elements.
23      */
24     private long _currentIndex = -1;
25
26     /**
27      * Setzt die Methode in den Anfangszustand zurueck.
28      */
29     @Override
30     public void reset() {
31         _structure = null;
32         _currentIndex = -1;
33     } // public void reset() ...
34
35     /**
36      * Fuehrt die aktuelle Methode fort.
37      * @param context Labeling-Context.
38      * @throws IllegalArgumentException Der angegebene Context ist null.
39      * @throws IllegalStateException Die Methode in einem ungueltigen Zustand.
40      */
41     @Override
42     public void execute(IntegerLabelingContext context) {
43         if (context == null) {
44             throw new IllegalArgumentException("context = null.");
45         } // if (context == null) ...
46         if (_structure == null) {
47             context.releaseMethodCycle(this);
48             context.removeMethod();
49         } else { // if (_structure != null) ...
50             long structureLength = _structure.getCount();
51             if (_currentIndex < 0) {
52                 if (structureLength <= 0) {
53                     context.releaseMethodCycle(this);
54                     context.removeMethod();
55                     return;
56                 } else { // if (structureLength > 0) ...
57                     _currentIndex = 0;
58                     IntegerLabelingMethod method = context.getLabelingMethod((Structure)
                           _structure.getElement(_currentIndex));
59                     context.pushMethod(method);
60                     return;
61                 } // if (structureLength > 0) ...
62             } else if (_currentIndex < structureLength - 1) {
63                 _currentIndex++;
64                 IntegerLabelingMethod method = context.getLabelingMethod((Structure) _structure
                       .getElement(_currentIndex));
65                 context.pushMethod(method);
66                 return;
67             } else { // if (_currentIndex >= structureLength - 1) ...
68                 context.releaseMethodCycle(this);
69                 context.removeMethod();
70                 return;
71             } // if (_currentIndex >= structureLength - 1) ...
72         } // if (_structure != null) ...
73     } // public void execute(IntegerLabelingContext context) ...
74 } // class IntegerLabelingMethodCycle extends IntegerLabelingMethod ...
```

Listing 90: Klasse *at.techmath.boltzmann.sampler.IntegerLabelingMethodOne*

```
1  package at.techmath.boltzmann.sampler;
2
3  /**
4   * Methode, die Integer-Label auf One-Strukturen verteilt.
5   * @author Stefan Schnabl (e0226245)
6   */
7  class IntegerLabelingMethodOne extends IntegerLabelingMethod {
8      /**
9       * Fuehrt die aktuelle Methode fort.
10      * @param context Labeling-Context.
11      * @throws IllegalArgumentException Der angegebene Context ist null.
12      * @throws IllegalStateException Die Methode in einem ungueltigen Zustand.
13      */
14     @Override
15     public void execute(IntegerLabelingContext context) {
16         if (context == null) {
17             throw new IllegalArgumentException("context = null.");
18         } // if (context == null) ...
19         context.releaseMethodOne(this);
20         context.removeMethod();
21     } // public void execute(IntegerLabelingContext context) ...
22 } // class IntegerLabelingMethodOne extends IntegerLabelingMethod ...
```

Listing 91: Klasse *at.techmath.boltzmann.sampler.IntegerLabelingMethodProduct*

```
1  package at.techmath.boltzmann.sampler;
2
3  /**
4   * Methode, die Integer-Label auf Produkt-Strukturen verteilt.
5   * @author Stefan Schnabl (e0226245)
6   */
7  class IntegerLabelingMethodProduct extends IntegerLabelingMethod {
8      /**
9       * Produkt-Struktur, die gerade gelabelt wird.
10      */
11     private StructureProduct _structure = null;
12
13     /**
14      * Setzt die Struktur, die gerade gelabelt wird.
15      * @param structure Struktur, die gerade gelabelt wird.
16      */
17     public void setStructure(StructureProduct structure) {
18         _structure = structure;
19     } // public void setStructure(StructureProduct structure) ...
20
21     /**
22      * Index des zuletzt gelabelten Faktors.
23      */
24     private long _currentIndex = -1;
25
26     /**
27      * Setzt die Methode in den Anfangszustand zurueck.
28      */
29     @Override
30     public void reset() {
31         _structure = null;
32         _currentIndex = -1;
33     } // public void reset() ...
34
35     /**
36      * Fuehrt die aktuelle Methode fort.
37      * @param context Labeling-Context.
38      * @throws IllegalArgumentException Der angegebene Context ist null.
39      * @throws IllegalStateException Die Methode in einem ungueltigen Zustand.
40      */
41     @Override
42     public void execute(IntegerLabelingContext context) {
43         if (context == null) {
44             throw new IllegalArgumentException("context = null.");
45         } // if (context == null) ...
46         if (_structure == null) {
47             context.releaseMethodProduct(this);
48             context.removeMethod();
49             return;
50         } else { // if (_structure != null) ...
51             long structureLength = _structure.getCount();
52             if (_currentIndex < 0) {
53                 if (structureLength <= 0) {
54                     context.releaseMethodProduct(this);
55                     context.removeMethod();
56                     return;
57                 } else { // if (structureLength > 0) ...
58                     _currentIndex = 0;
59                     IntegerLabelingMethod method = context.getLabelingMethod((Structure)
                           _structure.getElement(_currentIndex));
60                     context.pushMethod(method);
61                     return;
62                 } // if (structureLength > 0) ...
63             } else if (_currentIndex < structureLength - 1) {
64                 _currentIndex++;
65                 IntegerLabelingMethod method = context.getLabelingMethod((Structure) _structure
                       .getElement(_currentIndex));
66                 context.pushMethod(method);
67                 return;
68             } else { // if (_currentIndex >= structureLength - 1) ...
69                 context.releaseMethodProduct(this);
70                 context.removeMethod();
71                 return;
72             } // if (_currentIndex >= structureLength - 1) ...
73         } // if (_structure != null) ...
74     } // public void execute(IntegerLabelingContext context) ...
75 } // class IntegerLabelingMethodProduct extends IntegerLabelingMethod ...
```

Listing 92: Klasse *at.techmath.boltzmann.sampler.IntegerLabelingMethodSequence*

```
1  package at.techmath.boltzmann.sampler;
2
3  /**
4   * Methode, die Integer-Label auf Sequence-Strukturen verteilt.
5   * @author Stefan Schnabl (e0226245)
6   */
7  class IntegerLabelingMethodSequence extends IntegerLabelingMethod {
8      /**
9       * Sequence-Struktur, die gerade gelabelt wird.
10      */
11     private StructureSequence _structure = null;
12
13     /**
14      * Setzt die Struktur, die gerade gelabelt wird.
15      * @param structure Struktur, die gerade gelabelt wird.
16      */
17     public void setStructure(StructureSequence structure) {
18         _structure = structure;
19     } // public void setStructure(StructureSequence structure) ...
20
21     /**
22      * Index des zuletzt gelabelten Elements.
23      */
24     private long _currentIndex = -1;
25
26     /**
27      * Setzt die Methode in den Anfangszustand zurueck.
28      */
29     @Override
30     public void reset() {
31         _structure = null;
32         _currentIndex = -1;
33     } // public void reset() ...
34
35     /**
36      * Fuehrt die aktuelle Methode fort.
37      * @param context Labeling-Context.
38      * @throws IllegalArgumentException Der angegebene Context ist null.
39      * @throws IllegalStateException Die Methode in einem ungueltigen Zustand.
40      */
41     @Override
42     public void execute(IntegerLabelingContext context) {
43         if (context == null) {
44             throw new IllegalArgumentException("context = null.");
45         } // if (context == null) ...
46         if (_structure == null) {
47             context.releaseMethodSequence(this);
48             context.removeMethod();
49         } else { // if (_structure != null) ...
50             long structureLength = _structure.getCount();
51             if (_currentIndex < 0) {
52                 if (structureLength <= 0) {
53                     context.releaseMethodSequence(this);
54                     context.removeMethod();
55                     return;
56                 } else { // if (structureLength > 0) ...
57                     _currentIndex = 0;
58                     IntegerLabelingMethod method = context.getLabelingMethod((Structure)
                           _structure.getElement(_currentIndex));
59                     context.pushMethod(method);
60                     return;
61                 } // if (structureLength > 0) ...
62             } else if (_currentIndex < structureLength - 1) {
63                 _currentIndex++;
64                 IntegerLabelingMethod method = context.getLabelingMethod((Structure) _structure
                       .getElement(_currentIndex));
65                 context.pushMethod(method);
66                 return;
67             } else { // if (_currentIndex >= structureLength - 1) ...
68                 context.releaseMethodSequence(this);
69                 context.removeMethod();
70                 return;
71             } // if (_currentIndex >= structureLength - 1) ...
72         } // if (_structure != null) ...
73     } // public void execute(IntegerLabelingContext context) ...
74 } // class IntegerLabelingMethodSequence extends IntegerLabelingMethod ...
```

Listing 93: Klasse *at.techmath.boltzmann.sampler.IntegerLabelingMethodSet*

```
1  package at.techmath.boltzmann.sampler;
2
3  /**
4   * Methode, die Integer-Label auf Set-Strukturen verteilt.
5   * @author Stefan Schnabl (e0226245)
6   */
7  class IntegerLabelingMethodSet extends IntegerLabelingMethod {
8      /**
9       * Set-Struktur, die gerade gelabelt wird.
10      */
11     private StructureSet _structure = null;
12
13     /**
14      * Setzt die Struktur, die gerade gelabelt wird.
15      * @param structure Struktur, die gerade gelabelt wird.
16      */
17     public void setStructure(StructureSet structure) {
18         _structure = structure;
19     } // public void setStructure(StructureSet structure) ...
20
21     /**
22      * Index des zuletzt gelabelten Elements.
23      */
24     private long _currentIndex = -1;
25
26     /**
27      * Setzt die Methode in den Anfangszustand zurueck.
28      */
29     @Override
30     public void reset() {
31         _structure = null;
32         _currentIndex = -1;
33     } // public void reset() ...
34
35     /**
36      * Fuehrt die aktuelle Methode fort.
37      * @param context Labeling-Context.
38      * @throws IllegalArgumentException Der angegebene Context ist null.
39      * @throws IllegalStateException Die Methode in einem ungueltigen Zustand.
40      */
41     @Override
42     public void execute(IntegerLabelingContext context) {
43         if (context == null) {
44             throw new IllegalArgumentException("context = null.");
45         } // if (context == null) ...
46         if (_structure == null) {
47             context.releaseMethodSet(this);
48             context.removeMethod();
49         } else { // if (_structure != null) ...
50             long structureLength = _structure.getCount();
51             if (_currentIndex < 0) {
52                 if (structureLength <= 0) {
53                     context.releaseMethodSet(this);
54                     context.removeMethod();
55                     return;
56                 } else { // if (structureLength > 0) ...
57                     _currentIndex = 0;
58                     IntegerLabelingMethod method = context.getLabelingMethod((Structure)
                           _structure.getElement(_currentIndex));
59                     context.pushMethod(method);
60                     return;
61                 } // if (structureLength > 0) ...
62             } else if (_currentIndex < structureLength - 1) {
63                 _currentIndex++;
64                 IntegerLabelingMethod method = context.getLabelingMethod((Structure) _structure
                       .getElement(_currentIndex));
65                 context.pushMethod(method);
66                 return;
67             } else { // if (_currentIndex >= structureLength - 1) ...
68                 context.releaseMethodSet(this);
69                 context.removeMethod();
70                 return;
71             } // if (_currentIndex >= structureLength - 1) ...
72         } // if (_structure != null) ...
73     } // public void execute(IntegerLabelingContext context) ...
74 } // class IntegerLabelingMethodSet extends IntegerLabelingMethod ...
```

Listing 94: Klasse *at.techmath.boltzmann.sampler.IntegerLabelingMethodSingleton*

```
1  package at.techmath.boltzmann.sampler;
2
3  /**
4   * Methode, die Integer-Label auf Singleton-Strukturen verteilt.
5   * @author Stefan Schnabl (e0226245)
6   */
7  class IntegerLabelingMethodSingleton extends IntegerLabelingMethod {
8      /**
9       * Singleton-Struktur, die gerade gelabelt wird.
10      */
11     private StructureIntegerSingleton _structure = null;
12
13     /**
14      * Setzt die Struktur, die gerade gelabelt wird.
15      * @param structure Struktur, die gerade gelabelt wird.
16      */
17     public void setStructure(StructureIntegerSingleton structure) {
18         _structure = structure;
19     } // public void setStructure(StructureIntegerSingleton structure) ...
20
21     /**
22      * Setzt die Methode in den Anfangszustand zurueck.
23      */
24     @Override
25     public void reset() {
26         _structure = null;
27     } // public void reset() ...
28
29     /**
30      * Fuehrt die aktuelle Methode fort.
31      * @param context Labeling-Context.
32      * @throws IllegalArgumentException Der angegebene Context ist null.
33      * @throws IllegalStateException Die Methode in einem ungueltigen Zustand.
34      */
35     @Override
36     public void execute(IntegerLabelingContext context) {
37         if (context == null) {
38             throw new IllegalArgumentException("context = null.");
39         } // if (context == null) ...
40         if (_structure == null) {
41             context.releaseMethodSingleton(this);
42             context.removeMethod();
43         } else { // if (_structure != null) ...
44             _structure.setNumber(context.getNextAtom());
45             context.releaseMethodSingleton(this);
46             context.removeMethod();
47         } // if (_structure != null) ...
48     } // public void execute(IntegerLabelingContext context) ...
49 } // class IntegerLabelingMethodSingleton extends IntegerLabelingMethod ...
```

Listing 95: Klasse *at.techmath.boltzmann.sampler.IntegerLabelingMethodSum*

```
1  package at.techmath.boltzmann.sampler;
2
3  /**
4   * Methode, die Integer-Label auf Summen-Strukturen verteilt.
5   * @author Stefan Schnabl (e0226245)
6   */
7  class IntegerLabelingMethodSum extends IntegerLabelingMethod {
8      /**
9       * Summen-Struktur, die gerade gelabelt wird.
10      */
11     private StructureSum _structure = null;
12
13     /**
14      * Setzt die Struktur, die gerade gelabelt wird.
15      * @param structure Struktur, die gerade gelabelt wird.
16      */
17     public void setStructure(StructureSum structure) {
18         _structure = structure;
19     } // public void setStructure(StructureSum structure) ...
20
21     /**
22      * Wurde die untergeordnete Struktur schon gelabelt?
23      */
24     private boolean _labeled = false;
25
26     /**
27      * Setzt die Methode in den Anfangszustand zurueck.
28      */
29     @Override
30     public void reset() {
31         _structure = null;
32         _labeled = false;
33     } // public void reset() ...
34
35     /**
36      * Fuehrt die aktuelle Methode fort.
37      * @param context Labeling-Context.
38      * @throws IllegalArgumentException Der angegebene Context ist null.
39      * @throws IllegalStateException Die Methode in einem ungueltigen Zustand.
40      */
41     @Override
42     public void execute(IntegerLabelingContext context) {
43         if (context == null) {
44             throw new IllegalArgumentException("context = null.");
45         } // if (context == null) ...
46         if (_structure == null) {
47             context.releaseMethodSum(this);
48             context.removeMethod();
49             return;
50         } else { // if (_structure != null) ...
51             if (!_labeled) {
52                 IntegerLabelingMethod method = context.getLabelingMethod((Structure) _structure
                       .getStructure());
53                 context.pushMethod(method);
54                 _labeled = true;
55                 return;
56             } else { // if (_labeled) ...
57                 context.releaseMethodSum(this);
58                 context.removeMethod();
59                 return;
60             } // if (_labeled) ...
61         } // if (_structure != null) ...
62     } // public void execute(IntegerLabelingContext context) ...
63 } // class IntegerLabelingMethodSum extends IntegerLabelingMethod ...
```

Listing 96: Interface *at.techmath.boltzmann.sampler.ISamplingResult*

```
1  package at.techmath.boltzmann.sampler;
2
3  /**
4   * Repraesentiert das Ergebnis eines Sampling-Prozesses.
5   * @author Stefan Schnabl (e0226245)
6   */
7  public interface ISamplingResult {
8      /**
9       * Gibt die Struktur, die gesampelt wurde, zurueck.
10      * @return Struktur, die gesampelt wurde.
11      */
12     IStructure getStructure();
13
14     /**
15      * Gibt die Groesse der gesampelten Struktur zurueck.
16      * @return Groesse der gesampelten Struktur.
17      */
18     long getSize();
19 } // public interface ISamplingResult ...
```

Listing 97: Interface *at.techmath.boltzmann.sampler.IStringLabel*

```
1  package at.techmath.boltzmann.sampler;
2
3  /**
4   * Repraesentiert ein Text-Label.
5   * @author Stefan Schnabl (e0226245)
6   */
7  public interface IStringLabel extends ILabel {
8      /**
9       * Gibt den Text des Labels zurueck.
10      * @return Text des Labels.
11      */
12     String getText();
13 } // public interface IStringLabel extends ILabel ...
```

Listing 98: Interface *at.techmath.boltzmann.sampler.IStructure*

```
1 package at.techmath.boltzmann.sampler;
2
3 /**
4  * Repraesentiert eine Struktur einer Spezies.
5  * @author Stefan Schnabl (e0226245)
6  */
7 public interface IStructure {
8 } // public interface IStructure ...
```

Listing 99: Interface *at.techmath.boltzmann.sampler.IStructureCycle*

```
1  package at.techmath.boltzmann.sampler;
2
3  /**
4   * Repraesentiert eine Struktur der Cycle-Spezies.
5   * @author Stefan Schnabl (e0226245)
6   */
7  public interface IStructureCycle extends IStructure {
8      /**
9       * Gibt die Anzahl der Elemente des Zyklus zurueck.
10      * @return Anzahl der Elemente des Zyklus.
11      */
12     long getCount();
13
14     /**
15      * Gibt das Element mit dem angegebenen Index zurueck.
16      * @param index Index des gewuenschten Elements.
17      * @return Element mit dem angegebenen Index.
18      * @throws IndexOutOfBoundsException Der angegebene
19      *   Index ist ausserhalb des gueltigen Bereichs.
20      */
21     IStructure getElement(long index);
22 } // public interface IStructureCycle extends IStructure ...
```

Listing 100: Interface *at.techmath.boltzmann.sampler.IStructureIntegerSingleton*

```
1  package at.techmath.boltzmann.sampler;
2
3  /**
4   * Repraesentiert eine Singleton-Struktur auf der
5   * Menge der natuerlichen Zahlen.
6   * @author Stefan Schnabl (e0226245)
7   */
8  public interface IStructureIntegerSingleton extends IStructureSingleton {
9      /**
10      * Gibt die Nummer der Struktur zurueck.
11      * @return Nummer der Struktur.
12      */
13     long getNumber();
14 } // public interface IStructureIntegerSingleton extends IStructureSingleton ...
```

Listing 101: Interface *at.techmath.boltzmann.sampler.IStructureOne*

```
1 package at.techmath.boltzmann.sampler;
2
3 /**
4  * Repraesentiert eine Struktur der Spezies der Charakteristik
5  * der leeren Menge.
6  * @author Stefan Schnabl (e0226245)
7  */
8 public interface IStructureOne extends IStructure {
9 } // public interface IStructureOne extends IStructure ...
```

Listing 102: Interface *at.techmath.boltzmann.sampler.IStructureProduct*

```
1  package at.techmath.boltzmann.sampler;
2
3  /**
4   * Repraesentiert eine Struktur der Produkt-Spezies.
5   * @author Stefan Schnabl (e0226245)
6   */
7  public interface IStructureProduct extends IStructure {
8      /**
9       * Gibt die Anzahl der Faktoren zurueck.
10      * @return Anzahl der Faktoren.
11      */
12     long getCount();
13
14     /**
15      * Gibt das Element mit dem angegebenen Index zurueck.
16      * @param index Index des gewuenschten Elements.
17      * @return Element mit dem angegebenen Index.
18      * @throws IndexOutOfBoundsException Der angegebene
19      *   Index ist ausserhalb des gueltigen Bereichs.
20      */
21     IStructure getElement(long index);
22 } // public interface IStructureProduct extends IStructure ...
```

Listing 103: Interface *at.techmath.boltzmann.sampler.IStructureSequence*

```
1  package at.techmath.boltzmann.sampler;
2
3  /**
4   * Repraesentiert eine Struktur der Sequence-Spezies.
5   * @author Stefan Schnabl (e0226245)
6   */
7  public interface IStructureSequence extends IStructure {
8      /**
9       * Gibt die Anzahl der Elemente der Folge zurueck.
10      * @return Anzahl der Elemente der Folge.
11      */
12     long getCount();
13
14     /**
15      * Gibt das Element mit dem angegebenen Index zurueck.
16      * @param index Index des gewuenschten Elements.
17      * @return Element mit dem angegebenen Index.
18      * @throws IndexOutOfBoundsException Der angegebene
19      *   Index ist ausserhalb des gueltigen Bereichs.
20      */
21     IStructure getElement(long index);
22 } // public interface IStructureSequence extends IStructure ...
```

Listing 104: Interface *at.techmath.boltzmann.sampler.IStructureSet*

```
1  package at.techmath.boltzmann.sampler;
2
3  /**
4   * Repraesentiert eine Struktur der Set-Spezies.
5   * @author Stefan Schnabl (e0226245)
6   */
7  public interface IStructureSet extends IStructure {
8      /**
9       * Gibt die Anzahl der Elemente der Menge zurueck.
10      * @return Anzahl der Elemente der Menge.
11      */
12     long getCount();
13
14     /**
15      * Gibt das Element mit dem angegebenen Index zurueck.
16      * @param index Index des gewuenschten Elements.
17      * @return Element mit dem angegebenen Index.
18      * @throws IndexOutOfBoundsException Der angegebene
19      *   Index ist ausserhalb des gueltigen Bereichs.
20      */
21     IStructure getElement(long index);
22 } // public interface IStructureSet extends IStructure ...
```

Listing 105: Interface *at.techmath.boltzmann.sampler.IStructureSingleton*

```
1  package at.techmath.boltzmann.sampler;
2
3  /**
4   * Repraesentiert eine Struktur der Singleton-Spzeies.
5   * @author Stefan Schnabl (e0226245)
6   */
7  public interface IStructureSingleton extends IStructure {
8      /**
9       * Gibt das Element der Grundmenge zurueck,
10      * das durch die Struktur repraesentiert wird.
11      * @return Element der Grundmenge, das durch die
12      *   Struktur repraesentiert wird.
13      */
14     Object getAtom();
15 } // public interface IStructureSingleton extends IStructure ...
```

Listing 106: Interface *at.techmath.boltzmann.sampler.IStructureSum*

```
1  package at.techmath.boltzmann.sampler;
2
3  /**
4   * Repraesentiert eine Struktur der Summen-Spezies.
5   * @author Stefan Schnabl (e0226245)
6   */
7  public interface IStructureSum extends IStructure {
8      /**
9       * Gibt die Struktur, die vom Summen-Sampler gesampelt wurde, zurueck.
10      * @return Struktur, die vom Summen-Sampler gesampelt wurde.
11      */
12     IStructure getStructure();
13 } // public interface IStructureSum extends IStructure ...
```

Listing 107: Klasse *at.techmath.boltzmann.sampler.Label*

```
1  package at.techmath.boltzmann.sampler;
2
3  /**
4   * Basisklasse der Labels.
5   * @author Stefan Schnabl (e0226245)
6   */
7  abstract class Label implements ILabel {
8      /**
9       * Erzeugt aus dem uebergebenen Label der Spezifikation
10      * eine neue Instanz der entsprechenden Unterklasse.
11      * @return Neue Instanz der entsprechenden Unterklasse des angegebenen Spezifikations-Label
            .
12      * @throws IllegalArgumentException Wird geworfen, wenn das uebergebene Label null
13      *   oder von einem nicht unterstuetzten Typ ist.
14      */
15     public static Label createFromSpecificationLabel(at.techmath.boltzmann.specification.ILabel
            label) {
16         if (label == null) {
17             throw new IllegalArgumentException("label = null.");
18         } // if (label == null) ...
19         if (label instanceof at.techmath.boltzmann.specification.IIntegerLabel) {
20             return new IntegerLabel(((at.techmath.boltzmann.specification.IIntegerLabel) label)
                    .getNumber());
21         } else if (label instanceof at.techmath.boltzmann.specification.IStringLabel) {
22             return new StringLabel(((at.techmath.boltzmann.specification.IStringLabel) label).
                    getText());
23         } else {
24             throw new IllegalArgumentException("label hat nicht unterstuetzte Klasse.");
25         }
26     } // public static Label createFromSpecificationLabel(at.techmath.boltzmann.specification.
           ILabel label) ...
27 } // abstract class Label implements ILabel ...
```

Listing 108: Klasse *at.techmath.boltzmann.sampler.LabeledStructureCycle*

```
1  package at.techmath.boltzmann.sampler;
2
3  /**
4   * Struktur der Cycle-Spezies, die mit einem Label versehen ist.
5   * @author Stefan Schnabl (e0226245)
6   */
7  class LabeledStructureCycle extends StructureCycle implements ILabeledStructure {
8      /**
9       * Erzeugt eine neue Instanz.
10      * @param elements Elemente der Struktur.
11      * @throws IllegalArgumentException elements ist null.
12      */
13     public LabeledStructureCycle(Structure[] elements) {
14         super(elements);
15     } // public LabeledStructureCycle(Structure[] elements) ...
16
17     /**
18      * Label, mit dem die Struktur versehen ist.
19      */
20     private Label _label = null;
21
22     /**
23      * Setzt das Label, mit dem die Struktur versehen ist.
24      * @param label Label, mit dem die Struktur versehen ist.
25      */
26     protected void setLabel(Label label) {
27         _label = label;
28     } // protected void setLabel(Label label) ...
29
30     /**
31      * Gibt das Label zurueck, mit dem die Struktur versehen ist.
32      * @return Label, mit dem die Struktur versehen ist.
33      */
34     public ILabel getLabel() {
35         return _label;
36     } // public ILabel getLabel() ...
37 } // class LabeledStructureCycle extends StructureCycle implements ILabeledStructure ...
```

Listing 109: Klasse *at.techmath.boltzmann.sampler.LabeledStructureIntegerSingleton*

```
1  package at.techmath.boltzmann.sampler;
2
3  /**
4   * Struktur der Singleton-Spezies auf der Menge
5   * der natuerlichen Zahlen, die mit einem Label versehen ist.
6   * @author Stefan Schnabl (e0226245)
7   */
8  class LabeledStructureIntegerSingleton extends StructureIntegerSingleton implements
       ILabeledStructure {
9      /**
10      * Label, mit dem die Struktur versehen ist.
11      */
12     private Label _label = null;
13
14     /**
15      * Setzt das Label, mit dem die Struktur versehen ist.
16      * @param label Label, mit dem die Struktur versehen ist.
17      */
18     protected void setLabel(Label label) {
19         _label = label;
20     } // protected void setLabel(Label label) ...
21
22     /**
23      * Gibt das Label zurueck, mit dem die Struktur versehen ist.
24      * @return Label, mit dem die Struktur versehen ist.
25      */
26     public ILabel getLabel() {
27         return _label;
28     } // public ILabel getLabel() ...
29 } // class LabeledStructureIntegerSingleton extends StructureIntegerSingleton implements
       ILabeledStructure ...
```

Listing 110: Klasse *at.techmath.boltzmann.sampler.LabeledStructureOne*

```
1  package at.techmath.boltzmann.sampler;
2
3  /**
4   * Struktur der Spezies der Charakteristik der leeren Menge, die mit einem Label versehen ist.
5   * @author Stefan Schnabl (e0226245)
6   */
7  class LabeledStructureOne extends StructureOne implements ILabeledStructure {
8      /**
9       * Label, mit dem die Struktur versehen ist.
10      */
11     private Label _label = null;
12
13     /**
14      * Setzt das Label, mit dem die Struktur versehen ist.
15      * @param label Label, mit dem die Struktur versehen ist.
16      */
17     protected void setLabel(Label label) {
18         _label = label;
19     } // protected void setLabel(Label label) ...
20
21     /**
22      * Gibt das Label zurueck, mit dem die Struktur versehen ist.
23      * @return Label, mit dem die Struktur versehen ist.
24      */
25     public ILabel getLabel() {
26         return _label;
27     } // public ILabel getLabel() ...
28 } // class LabeledStructureOne extends StructureOne implements ILabeledStructure ...
```

Listing 111: Klasse *at.techmath.boltzmann.sampler.LabeledStructureProduct*

```
1  package at.techmath.boltzmann.sampler;
2
3  /**
4   * Struktur der Produkt-Spezies, die mit einem Label versehen ist.
5   * @author Stefan Schnabl (e0226245)
6   */
7  class LabeledStructureProduct extends StructureProduct implements ILabeledStructure {
8      /**
9       * Erzeugt eine neue Instanz.
10      * @param elements Faktoren der Struktur.
11      * @throws System.IllegalArgumentException elements ist null.
12      */
13     public LabeledStructureProduct(Structure[] elements) {
14         super(elements);
15     } // public LabeledStructureProduct(Structure[] elements) ...
16
17     /**
18      * Label, mit dem die Struktur versehen ist.
19      */
20     private Label _label = null;
21
22     /**
23      * Setzt das Label, mit dem die Struktur versehen ist.
24      * @param label Label, mit dem die Struktur versehen ist.
25      */
26     protected void setLabel(Label label) {
27         _label = label;
28     } // protected void setLabel(Label label) ...
29
30     /**
31      * Gibt das Label zurueck, mit dem die Struktur versehen ist.
32      * @return Label, mit dem die Struktur versehen ist.
33      */
34     public ILabel getLabel() {
35         return _label;
36     } // public ILabel getLabel() ...
37 } // class LabeledStructureProduct extends StructureProduct implements ILabeledStructure ...
```

Listing 112: Klasse *at.techmath.boltzmann.sampler.LabeledStructureSequence*

```
1  package at.techmath.boltzmann.sampler;
2
3  /**
4   * Struktur der Sequence-Spezies, die mit einem Label versehen ist.
5   * @author Stefan Schnabl (e0226245)
6   */
7  class LabeledStructureSequence extends StructureSequence implements ILabeledStructure {
8      /**
9       * Erzeugt eine neue Instanz.
10      * @param elements Elemente der Struktur.
11      * @throws IllegalArgumentException elements ist null.
12      */
13     public LabeledStructureSequence(Structure[] elements) {
14         super(elements);
15     } // public LabeledStructureSequence(Structure[] elements) ...
16
17     /**
18      * Label, mit dem die Struktur versehen ist.
19      */
20     private Label _label = null;
21
22     /**
23      * Setzt das Label, mit dem die Struktur versehen ist.
24      * @param label Label, mit dem die Struktur versehen ist.
25      */
26     protected void setLabel(Label label) {
27         _label = label;
28     } // protected void setLabel(Label label) ...
29
30     /**
31      * Gibt das Label zurueck, mit dem die Struktur versehen ist.
32      * @return Label, mit dem die Struktur versehen ist.
33      */
34     public ILabel getLabel() {
35         return _label;
36     } // public ILabel getLabel() ...
37 } // class LabeledStructureSequence extends StructureSequence implements ILabeledStructure ...
```

Listing 113: Klasse *at.techmath.boltzmann.sampler.LabeledStructureSet*

```
1  package at.techmath.boltzmann.sampler;
2
3  /**
4   * Struktur der Set-Spezies, die mit einem Label versehen ist.
5   * @author Stefan Schnabl (e0226245)
6   */
7  class LabeledStructureSet extends StructureSet implements ILabeledStructure {
8      /**
9       * Erzeugt eine neue Instanz.
10      * @param elements Elemente der Struktur.
11      * @throws IllegalArgumentException elements ist null.
12      */
13     public LabeledStructureSet(Structure[] elements) {
14         super(elements);
15     } // public LabeledStructureSet(Structure[] elements) ...
16
17     /**
18      * Label, mit dem die Struktur versehen ist.
19      */
20     private Label _label = null;
21
22     /**
23      * Setzt das Label, mit dem die Struktur versehen ist.
24      * @param label Label, mit dem die Struktur versehen ist.
25      */
26     protected void setLabel(Label label) {
27         _label = label;
28     } // protected void setLabel(Label label) ...
29
30     /**
31      * Gibt das Label zurueck, mit dem die Struktur versehen ist.
32      * @return Label, mit dem die Struktur versehen ist.
33      */
34     public ILabel getLabel() {
35         return _label;
36     } // public ILabel getLabel() ...
37 } // class LabeledStructureSet extends StructureSet implements ILabeledStructure ...
```

Listing 114: Klasse *at.techmath.boltzmann.sampler.LabeledStructureSingleton*

```
1  package at.techmath.boltzmann.sampler;
2
3  /**
4   * Struktur der Singleton-Spezies, die mit einem Label versehen ist.
5   * @author Stefan Schnabl (e0226245)
6   */
7  class LabeledStructureSingleton extends StructureSingleton implements ILabeledStructure {
8      /**
9       * Label, mit dem die Struktur versehen ist.
10      */
11     private Label _label = null;
12
13     /**
14      * Setzt das Label, mit dem die Struktur versehen ist.
15      * @param label Label, mit dem die Struktur versehen ist.
16      */
17     protected void setLabel(Label label) {
18         _label = label;
19     } // protected void setLabel(Label label) ...
20
21     /**
22      * Gibt das Label zurueck, mit dem die Struktur versehen ist.
23      * @return Label, mit dem die Struktur versehen ist.
24      */
25     public ILabel getLabel() {
26         return _label;
27     } // public ILabel getLabel() ...
28 } // class LabeledStructureSingleton extends StructureSingleton implements ILabeledStructure
      ...
```

Listing 115: Klasse *at.techmath.boltzmann.sampler.LabeledStructureSum*

```
1  package at.techmath.boltzmann.sampler;
2
3  /**
4   * Struktur der Summen-Spezies, die mit einem Label versehen ist.
5   * @author Stefan Schnabl (e0226245)
6   */
7  public class LabeledStructureSum extends StructureSum implements ILabeledStructure {
8      /**
9       * Erzeugt eine neue Instanz.
10      * @param structure Struktur, die vom Summen-Sampler gesampelt wurde.
11      * @throws IllegalArgumentException structure ist null.
12      */
13     public LabeledStructureSum(Structure structure) {
14         super(structure);
15     } // public LabeledStructureSum(Structure structure) ...
16
17     /**
18      * Label, mit dem die Struktur versehen ist.
19      */
20     private Label _label = null;
21
22     /**
23      * Setzt das Label, mit dem die Struktur versehen ist.
24      * @param label Label, mit dem die Struktur versehen ist.
25      */
26     protected void setLabel(Label label) {
27         _label = label;
28     } // protected void setLabel(Label label) ...
29
30     /**
31      * Gibt das Label zurueck, mit dem die Struktur versehen ist.
32      * @return Label, mit dem die Struktur versehen ist.
33      */
34     public ILabel getLabel() {
35         return _label;
36     } // public ILabel getLabel() ...
37 } // public class LabeledStructureSum extends StructureSum implements ILabeledStructure ...
```

Listing 116: Klasse *at.techmath.boltzmann.sampler.LabelingContext*

```
1 package at.techmath.boltzmann.sampler;
2
3 /**
4  * Labeling-Context.
5  * @author Stefan Schnabl (e0226245)
6  */
7 class LabelingContext {
8 } // class LabelingContext ...
```

Listing 117: Klasse *at.techmath.boltzmann.sampler.LabelingMethod*

```
1  package at.techmath.boltzmann.sampler;
2
3  /**
4   * Basisklasse fuer Methoden, die die Atome der Grundmenge
5   * der Struktur verteilen.
6   * @author Stefan Schnabl (e0226245)
7   */
8  abstract class LabelingMethod {
9      /**
10      * Setzt die Methode in den Anfangszustand zurueck.
11      */
12     public void reset() {
13     } // public void reset() ...
14
15     /**
16      * Fuehrt die aktuelle Methode fort.
17      * @param context Labeling-Context.
18      * @throws IllegalArgumentException Der angegebene Context ist null.
19      * @throws IllegalStateException Die Methode in einem ungueltigen Zustand.
20      */
21     public abstract void execute(LabelingContext context);
22 } // class LabelingMethod ...
```

Listing 118: Klasse *at.techmath.boltzmann.sampler.SamplerMethod*

```
1  package at.techmath.boltzmann.sampler;
2
3  /**
4   * Basisklasse fuer eine Methode, die eine
5   * Struktur sampelt.
6   * @author Stefan Schnabl (e0226245)
7   */
8  abstract class SamplerMethod {
9      /**
10      * Label, mit dem die erzeugten Strukturen versehen werden sollen.
11      */
12     private Label _label = null;
13
14     /**
15      * Gibt das Label, mit dem die Strukuren versehen werden sollen, zurueck.
16      * @return Label, mit dem die Strukturen versehen werden sollen.
17      */
18     public Label getLabel() {
19         return _label;
20     } // public Label getLabel() ...
21
22     /**
23      * Setzt das Label, mit dem die Strukturen versehen werden sollen.
24      * @param label Label, mit dem die Strukturen versehen werden sollen.
25      */
26     public void setLabel(Label label) {
27         _label = label;
28     } // public void setLabel(Label label) ...
29
30     /**
31      * Setzt die Methode in den Anfangszustand zurueck.
32      */
33     public void reset() {
34         _label = null;
35     } // public void reset() ...
36
37     /**
38      * Fuehrt die aktuelle Methode fort.
39      * @param context Sampling-Context.
40      * @throws IllegalArgumentException Der angegebene Context ist null.
41      * @throws IllegalStateException Die Methode in einem ungueltigen Zustand.
42      */
43     public abstract void execute(SamplingContext context);
44 } // abstract class SamplerMethod ...
```

Listing 119: Klasse *at.techmath.boltzmann.sampler.SamplingContext*

```
1  package at.techmath.boltzmann.sampler;
2
3  import java.util.Stack;
4  import at.techmath.boltzmann.specification.ISpecification;
5  import at.techmath.boltzmann.oracle.IExponentialBoltzmannOracle;
6  import at.techmath.boltzmann.random.IUniformRandomGenerator;
7  import at.techmath.boltzmann.random.IGeometricRandomGenerator;
8  import at.techmath.boltzmann.random.IPoissonRandomGenerator;
9  import at.techmath.boltzmann.random.ILogarithmicRandomGenerator;
10
11 /**
12  * Sampling-Context.
13  * @author Stefan Schnabl (e0226245)
14  */
15 class SamplingContext {
16     /**
17      * Erzeugt einen neuen Sampling-Context zum angegebenen Boltzmann-Parameter.
18      * @param oracle Boltzmann-Orakel, das die EEF der Spezifikation auswerten kann.
19      * @param randomGenerator Zufallszahlengenerator.
20      * @param x Parameter des Boltzmann-Modells.
21      * @return Neuer Sampling-Context.
22      * @throws IllegalArgumentException Wird geworfen, wenn oracle, die Spezifikation oder
            einer der RandomGenerators null ist.
23      */
24     public static SamplingContext createContext(IExponentialBoltzmannOracle oracle,
25         IUniformRandomGenerator uniformRandomGenerator, IGeometricRandomGenerator
                geometricRandomGenerator,
26         IPoissonRandomGenerator poissonRandomGenerator, ILogarithmicRandomGenerator
                logarithmicRandomGenerator,
27         double x) {
28         if (oracle == null) {
29             throw new IllegalArgumentException("oracle = null.");
30         } // if (oracle == null) ...
31         if (uniformRandomGenerator == null) {
32             throw new IllegalArgumentException("uniformRandomGenerator = null.");
33         } // if (uniformRandomGenerator == null) ...
34         if (geometricRandomGenerator == null) {
35             throw new IllegalArgumentException("geometricRandomGenerator = null.");
36         } // if (geometricRandomGenerator == null) ...
37         if (poissonRandomGenerator == null) {
38             throw new IllegalArgumentException("poissonRandomGenerator = null.");
39         } // if (poissonRandomGenerator == null) ...
40         if (logarithmicRandomGenerator == null) {
41             throw new IllegalArgumentException("logarithmicRandomGenerator = null.");
42         } // if (logarithmicRandomGenerator == null) ...
43         ISpecification specification = oracle.getSpecification();
44         if (specification == null) {
45             throw new IllegalArgumentException("specification = null.");
46         } // if (specification == null) ...
47         // Ergebnis-Instanz erzeugen.
48         SamplingContext result = new SamplingContext();
49         result._uniformRandomGenerator = uniformRandomGenerator;
50         result._geometricRandomGenerator = geometricRandomGenerator;
51         result._poissonRandomGenerator = poissonRandomGenerator;
52         result._logarithmicRandomGenerator = logarithmicRandomGenerator;
53         // Anzahl der Spezies ermitteln.
54         int size = specification.getCount();
55         // Erzeugende Funktionen an der Stelle x auswerten und die Werte im Context speichern.
56         double[] eefValues = result._generatingFunctionValues = new double[size];
57         for (int c = 0; c < size; c++) {
58             eefValues[c] = oracle.evaluate(c, x);
59         } // for (int c = 0; c < size; c++) ...
60         result._boltzmannParameter = x;
61         return result;
62     } // public static SamplingContext createContext(IExponentialBoltzmannOracle oracle, ...)
            ...
63
64     /**
65      * Parameter, mit dem der Boltzmann-Sampler aufgerufen wird.
66      */
67     double _boltzmannParameter = 0.0D;
68
69     /**
70      * Gibt den Parameter, mit dem der Boltzmann-Sampler aufgerufen wird, zurueck.
71      * @return Parameter, mit dem der Boltzmann-Sampler aufgerufen wird.
72      */
73     public double getBoltzmannParameter() {
74         return _boltzmannParameter;
75     } // public double getBoltzmannParameter() ...
```

```
76
77      /**
78       * Werte der erzeugenden Funktionen.
79       */
80      double[] _generatingFunctionValues = new double[0];
81
82      /**
83       * Gibt den Wert der erzeugenden Funktion der Spezies
84       * mit dem angegebenen Index zurueck.
85       * @param index Index der Spezies.
86       * @return Wert der erzeugenden Funktion.
87       * @throws IndexOutOfBoundsException Der angegebene Index liegt
88       *   ausserhalb des gueltigen Bereichs.
89       */
90      public double getGeneratingFunctionValue(int index) {
91          if (index < 0 || index >= _generatingFunctionValues.length) {
92              throw new IndexOutOfBoundsException("Index ausserhalb des gueltigen Bereichs.");
93          } // if (index < 0 || index >= _generatingFunctionValues.length) ...
94          return _generatingFunctionValues[index];
95      } // public double getGeneratingFunctionValue(int index) ...
96
97      /**
98       * Stack der Sampling-Methoden, die gerade ausgefuehrt werden.
99       */
100     private Stack<SamplerMethod> _samplerMethodStack = new Stack<SamplerMethod>();
101
102     /**
103      * Gibt zurueck, ob der Sampling-Methoden-Stack leer ist.
104      * @return Ist der Sampling-Methoden-Stack leer?
105      */
106     public boolean isEmpty() {
107         return _samplerMethodStack.empty();
108     } // public boolean isEmpty() ...
109
110     /**
111      * Entfernt die zuletzt ausgefuehrte Methode.
112      */
113     public void removeMethod() {
114         _samplerMethodStack.pop();
115     } // public void removeMethod() ...
116
117     /**
118      * Legt die angegebene Methode auf den Methoden-Stack.
119      * @param method Methode, die auf den Methoden-Stack gelegt werden soll.
120      * @throws IllegalArgumentException Die angegebene Methode ist null.
121      */
122     public void pushMethod(SamplerMethod method) {
123         if (method == null) throw new IllegalArgumentException("method = null.");
124         _samplerMethodStack.push(method);
125     } // public void pushMethod(SamplerMethod method) ...
126
127     /**
128      * Gibt die aktuelle Methode des Methoden-Stacks zurueck.
129      * @return Aktuelle Methode des Methoden-Stacks.
130      */
131     public SamplerMethod peekMethod() {
132         return _samplerMethodStack.peek();
133     } // public SamplerMethod peekMethod() ...
134
135     /**
136      * Aktuell zurueckgegebene Struktur.
137      */
138     private Structure _result = null;
139
140     /**
141      * Gibt die aktuell zurueckgegebene Struktur zurueck.
142      * @return Aktuell zurueckgegebene Struktur.
143      */
144     public Structure getResult() {
145         return _result;
146     } // public Structure getResult() ...
147
148     /**
149      * Setzt die aktuell zurueckgegebene Struktur.
150      * @param result Aktuell zurueckgegebene Struktur.
151      */
152     public void setResult(Structure result) {
153         _result = result;
154     } // public void setResult(Structure result) ...
155
```

```
156     /**
157      * Aktuelle Groesse der Struktur.
158      */
159     private long _size = 0L;
160 
161     /**
162      * Gibt die aktuelle Groesse der Struktur zurueck.
163      * @return Aktuelle Groesse der Struktur.
164      */
165     public long getSize() {
166         return _size;
167     } // public long getSize() ...
168 
169     /**
170      * Erhoeht die aktuelle Groesse der Struktur um eins.
171      */
172     public void incrementSize() {
173         _size++;
174     } // public void incrementSize() ...
175 
176     /**
177      * Generator fuer auf [0,1) unabhaengig gleichverteilte Zufallsvariablen.
178      */
179     private IUniformRandomGenerator _uniformRandomGenerator;
180 
181     /**
182      * Generator fuer geometrisch verteilte Zufallsvariablen.
183      */
184     private IGeometricRandomGenerator _geometricRandomGenerator;
185 
186     /**
187      * Generator fuer Poisson-verteilte Zufallsvariablen.
188      */
189     private IPoissonRandomGenerator _poissonRandomGenerator;
190 
191     /**
192      * Generator fuer logarithmisch verteilte Zufallsvariablen.
193      */
194     private ILogarithmicRandomGenerator _logarithmicRandomGenerator;
195 
196     /**
197      * Gibt eine auf [0, 1) unabhaengig gleichverteilte Zufallszahl zurueck.
198      * @return Eine auf [0, 1) unabhaengig gleichverteilte Zufallszahl.
199      */
200     public double generateUniformRandomNumber() {
201         return _uniformRandomGenerator.getDouble();
202     } // public double generateUniformRandomNumber() ...
203 
204     /**
205      * Gibt eine unabhaengige Zufallszahl, die mit dem angegebenen
206      * Parameter geometrisch verteilt ist, zurueck.
207      * @param parameter Parameter der geometrischen Verteilung.
208      * @return Unabhaengige Zufallszahl, die mit dem angegebenen
209      *   Parameter geometrisch verteilt ist.
210      * @throws IllegalArgumentException Der Parameter ist ausserhalb des gueltigen Bereichs.
211      */
212     public int generateGeometricRandomNumber(double parameter) {
213         return _geometricRandomGenerator.generateGeometric(_uniformRandomGenerator, parameter);
214     } // public int generateGeometricRandomNumber(double parameter) ...
215 
216     /**
217      * Gibt eine unabhaengige Zufallszahl, die mit dem angegebenen
218      * Parameter Poisson-verteilt ist, zurueck.
219      * @param parameter Parameter der Poisson-Verteilung.
220      * @return Unabhaengige Zufallszahl, die mit dem angegebenen
221      *   Parameter Poisson-verteilt ist.
222      * @throws IllegalArgumentException Der Parameter ist ausserhalb des gueltigen Bereichs.
223      */
224     public int generatePoissonRandomNumber(double parameter) {
225         return _poissonRandomGenerator.generatePoisson(_uniformRandomGenerator, parameter);
226     } // public int generatePoissonRandomNumber(double parameter) ...
227 
228     /**
229      * Gibt eine unabhaengige Zufallszahl, die mit dem angegebenen
230      * Parameter logarithmisch verteilt ist, zurueck.
231      * @param parameter Parameter der logarithmischen Verteilung.
232      * @return Unabhaengige Zufallszahl, die mit dem angegebenen
233      *   Parameter logarithmisch verteilt ist.
234      * @throws IllegalArgumentException Der Parameter ist ausserhalb des gueltigen Bereichs.
235      */
```

```
236     public int generateLogarithmicRandomNumber(double parameter) {
237         return _logarithmicRandomGenerator.generateLogarithmic(_uniformRandomGenerator,
                parameter);
238     } // public int generateLogarithmicRandomNumber(double parameter) ...
239 } // class SamplingContext ...
```

Listing 120: Klasse *at.techmath.boltzmann.sampler.SamplingResult*

```
1  package at.techmath.boltzmann.sampler;
2
3  /**
4   * Standard-Implementierung des ISamplingResult-Interfaces.
5   * @author Stefan Schnabl (e0226245)
6   */
7  class SamplingResult implements ISamplingResult {
8      /**
9       * Erzeugt eine neue Instanz.
10      * @param structure Struktur, die gesampelt wurde.
11      * @param size Greosse der gesampelten Struktur.
12      * @throws IllegalArgumentException Wird geworfen, wenn structure = null ist oder size < 0.
13      */
14     public SamplingResult(Structure structure, long size) {
15         if (structure == null) {
16             throw new IllegalArgumentException("structure = null.");
17         } // if (structure == null) ...
18         if (size < 0L) {
19             throw new IllegalArgumentException("size < 0.");
20         } // if (size < 0L) ...
21         _structure = structure;
22         _size = size;
23     } // public SamplingResult(Structure structure, long size) ...
24
25     /**
26      * Struktur, die gesampelt wurde.
27      */
28     private Structure _structure;
29
30     /**
31      * Groesse der gesampelten Struktur.
32      */
33     private long _size;
34
35     /**
36      * Gibt die Struktur, die gesampelt wurde, zurueck.
37      * @return Struktur, die gesampelt wurde.
38      */
39     public IStructure getStructure() {
40         return _structure;
41     } // public IStructure getStructure() ...
42
43     /**
44      * Gibt die Groesse der gesampelten Struktur zurueck.
45      * @return Groesse der gesampelten Struktur.
46      */
47     public long getSize() {
48         return _size;
49     } // public long getSize() ...
50 } // class SamplingResult implements ISamplingResult ...
```

Listing 121: Klasse *at.techmath.boltzmann.sampler.StringLabel*

```
1  package at.techmath.boltzmann.sampler;
2
3  /**
4   * Text-Label.
5   * @author Stefan Schnabl (e0226245)
6   */
7  class StringLabel extends Label implements IStringLabel {
8      /**
9       * Erzeugt eine neue Instanz.
10      * @param text Text des Labels.
11      */
12     public StringLabel(String text) {
13         super();
14         if (text == null) {
15             _text = "";
16         } else { // if (text != null) ...
17             _text = text;
18         } // if (text != null) ...
19     } // public StringLabel(String text) ...
20
21     /**
22      * Text des Labels.
23      */
24     private String _text;
25
26     /**
27      * Gibt den Text des Labels zurueck.
28      * @return Text des Labels.
29      */
30     public String getText() {
31         return _text;
32     } // public String getText() ...
33 } // class StringLabel extends Label implements IStringLabel ...
```

Listing 122: Klasse *at.techmath.boltzmann.sampler.Structure*

```
1 package at.techmath.boltzmann.sampler;
2
3 /**
4  * Basisklasse fuer die Strukturen.
5  * @author Stefan Schnabl (e0226245)
6  */
7 abstract class Structure implements IStructure {
8 } // abstract class Structure implements IStructure ...
```

Listing 123: Klasse *at.techmath.boltzmann.sampler.StructureCycle*

```
1  package at.techmath.boltzmann.sampler;
2
3  /**
4   * Standard-Implementierung des IStructureCycle-Interfaces.
5   * @author Stefan Schnabl (e0226245)
6   */
7  class StructureCycle extends Structure implements IStructureCycle {
8      /**
9       * Erzeugt eine neue Instanz.
10      * @param elements Elemente der Struktur.
11      * @throws IllegalArgumentException elements ist null.
12      */
13     public StructureCycle(Structure[] elements) {
14         super();
15         if (elements == null) {
16             throw new IllegalArgumentException("elements = null.");
17         } // if (elements == null) ...
18         _elements = elements;
19     } // public StructureCycle(Structure[] elements) ...
20
21     /**
22      * Elemente des Zyklus.
23      */
24     private Structure[] _elements;
25
26     /**
27      * Gibt die Anzahl der Elemente des Zyklus zurueck.
28      * @return Anzahl der Elemente des Zyklus.
29      */
30     public long getCount() {
31         return _elements.length;
32     } // public long getCount() ...
33
34     /**
35      * Gibt das Element mit dem angegebenen Index zurueck.
36      * @param index Index des gewuenschten Elements.
37      * @return Element mit dem angegebenen Index.
38      * @throws IndexOutOfBoundsException Der angegebene
39      *   Index ist ausserhalb des gueltigen Bereichs.
40      */
41     public IStructure getElement(long index) {
42         if (index < 0L || index >= _elements.length) throw new IndexOutOfBoundsException("Index
                ausserhalb des gueltigen Bereichs.");
43         return _elements[(int) index];
44     } // public IStructure getElement(long index) ...
45 } // class StructureCycle extends Structure implements IStructureCycle ...
```

Listing 124: Klasse *at.techmath.boltzmann.sampler.StructureIntegerSingleton*

```
1  package at.techmath.boltzmann.sampler;
2
3  /**
4   * Standard-Implementierung des IStructureIntegerSingleton-Interfaces.
5   * @author Stefan Schnabl (e0226245)
6   */
7  class StructureIntegerSingleton extends Structure implements IStructureIntegerSingleton {
8      /**
9       * Nummer der Struktur.
10      */
11     private long _number = 0L;
12
13     /**
14      * Setzt die Nummer der Struktur.
15      * @param value Nummer der Struktur.
16      */
17     protected void setNumber(long value) {
18         _number = value;
19     } // protected void setNumber(long value) ...
20
21     /**
22      * Gibt die Nummer der Struktur zurueck.
23      * @return Nummer der Struktur.
24      */
25     public long getNumber() {
26         return _number;
27     } // public long getNumber() ...
28
29     /**
30      * Gibt das Element der Grundmenge zurueck,
31      * das durch die Struktur repraesentiert wird.
32      * @return Element der Grundmenge, das durch die
33      *   Struktur repraesentiert wird.
34      */
35     public Object getAtom() {
36         return _number;
37     } // public Object getAtom() ...
38 } // class StructureIntegerSingleton extends Structure implements IStructureIntegerSingleton
      ...
```

Listing 125: Klasse *at.techmath.boltzmann.sampler.StructureOne*

```
1 package at.techmath.boltzmann.sampler;
2
3 /**
4  * Standard-Implementierung des IStructureOne-Interfaces.
5  * @author Stefan Schnabl (e0266245)
6  */
7 class StructureOne extends Structure implements IStructureOne {
8 } // class StructureOne extends Structure implements IStructureOne ...
```

Listing 126: Klasse *at.techmath.boltzmann.sampler.StructureProduct*

```
1  package at.techmath.boltzmann.sampler;
2
3  /**
4   * Standard-Implementierung des IStructureProduct-Interfaces.
5   * @author Stefan Schnabl (e0226245)
6   */
7  class StructureProduct extends Structure implements IStructureProduct {
8      /**
9       * Erzeugt eine neue Instanz.
10      * @param elements Faktoren der Struktur.
11      * @throws IllegalArgumentException elements ist null.
12      */
13     public StructureProduct(Structure[] elements) {
14         super();
15         if (elements == null) {
16             throw new IllegalArgumentException("elements = null.");
17         } // if (elements == null) ...
18         _elements = elements;
19     } // public StructureProduct(Structure[] elements) ...
20
21     /**
22      * Faktoren des Produkts.
23      */
24     private Structure[] _elements;
25
26     /**
27      * Gibt die Anzahl der Faktoren zurueck.
28      * @return Anzahl der Faktoren.
29      */
30     public long getCount() {
31         return _elements.length;
32     } // public long getCount() ...
33
34     /**
35      * Gibt das Element mit dem angegebenen Index zurueck.
36      * @param index Index des gewuenschten Elements.
37      * @return Element mit dem angegebenen Index.
38      * @throws IndexOutOfBoundsException Der angegebene
39      *   Index ist ausserhalb des gueltigen Bereichs.
40      */
41     public IStructure getElement(long index) {
42         if (index < 0L || index >= _elements.length) throw new IndexOutOfBoundsException("Index
                ausserhalb des gueltigen Bereichs.");
43         return _elements[(int) index];
44     } // public IStructure getElement(long index) ...
45 }  // class StructureProduct extends Structure implements IStructureProduct ...
```

Listing 127: Klasse *at.techmath.boltzmann.sampler.StructureSequence*

```
1  package at.techmath.boltzmann.sampler;
2
3  /**
4   * Basis-Implementierung des IStructureSequence-Interfaces.
5   * @author Stefan Schnabl (e0226245)
6   */
7  class StructureSequence extends Structure implements IStructureSequence {
8      /**
9       * Erzeugt eine neue Instanz.
10      * @param elements Elemente der Struktur.
11      * @throws System.IllegalArgumentException elements ist null.
12      */
13     public StructureSequence(Structure[] elements) {
14         super();
15         if (elements == null) {
16             throw new IllegalArgumentException("elements = null.");
17         } // if (elements == null) ...
18         _elements = elements;
19     } // public StructureSequence(Structure[] elements) ...
20
21     /**
22      * Elemente der Folge.
23      */
24     private Structure[] _elements;
25
26     /**
27      * Gibt die Anzahl der Elemente der Folge zurueck.
28      * @return Anzahl der Elemente der Folge.
29      */
30     public long getCount() {
31         return _elements.length;
32     } // public long getCount() ...
33
34     /**
35      * Gibt das Element mit dem angegebenen Index zurueck.
36      * @param index Index des gewuenschten Elements.
37      * @return Element mit dem angegebenen Index.
38      * @throws IndexOutOfBoundsException Der angegebene
39      *   Index ist ausserhalb des gueltigen Bereichs.
40      */
41     public IStructure getElement(long index) {
42         if (index < 0L || index >= _elements.length) throw new IndexOutOfBoundsException("Index
                ausserhalb des gueltigen Bereichs.");
43         return _elements[(int) index];
44     } // public IStructure getElement(long index) ...
45 } // class StructureSequence extends Structure implements IStructureSequence ...
```

Listing 128: Klasse *at.techmath.boltzmann.sampler.StructureSet*

```
1  package at.techmath.boltzmann.sampler;
2
3  /**
4   * Standard-Implementierung des IStructureSet-Interfaces.
5   * @author Stefan Schnabl (e0226245)
6   */
7  class StructureSet extends Structure implements IStructureSet {
8      /**
9       * Erzeugt eine neue Instanz.
10      * @param elements Elemente der Struktur.
11      * @throws IllegalArgumentException elements ist null.
12      */
13     public StructureSet(Structure[] elements) {
14         super();
15         if (elements == null) {
16             throw new IllegalArgumentException("elements = null.");
17         } // if (elements == null) ...
18         _elements = elements;
19     } // public StructureSet(Structure[] elements) ...
20
21     /**
22      * Elemente der Menge.
23      */
24     private Structure[] _elements;
25
26     /**
27      * Gibt die Anzahl der Elemente der Menge zurueck.
28      * @return Anzahl der Elemente der Menge.
29      */
30     public long getCount() {
31         return _elements.length;
32     } // public long getCount() ...
33
34
35     /**
36      * Gibt das Element mit dem angegebenen Index zurueck.
37      * @param index Index des gewuenschten Elements.
38      * @return Element mit dem angegebenen Index.
39      * @throws IndexOutOfBoundsException Der angegebene
40      *   Index ist ausserhalb des gueltigen Bereichs.
41      */
42     public IStructure getElement(long index) {
43         if (index < 0L || index >= _elements.length) throw new IndexOutOfBoundsException("Index
               ausserhalb des gueltigen Bereichs.");
44         return _elements[(int) index];
45     } // public IStructure getElement(long index) ...
46 } // class StructureSet extends Structure implements IStructureSet ...
```

Listing 129: Klasse *at.techmath.boltzmann.sampler.StructureSingleton*

```
1  package at.techmath.boltzmann.sampler;
2
3  /**
4   * Strukur der Singleton-Spezies, die ein Atom der Grundmenge
5   * repraesentiert.
6   * @author Stefan Schnabl (e0226245)
7   */
8  class StructureSingleton extends Structure implements IStructureSingleton {
9      /**
10      * Atom, das von der Struktur repraesentiert wird.
11      */
12     private Object _atom = null;
13
14     /**
15      * Setzt das Atom, das von der Struktur repraesentiert wird.
16      * @param value Atom, das von der Struktur repraesentiert wird.
17      */
18     protected void setAtom(Object value) {
19         _atom = value;
20     } // protected void setAtom(Object value) ...
21
22     /**
23      * Gibt das Element der Grundmenge zurueck,
24      * das durch die Struktur repraesentiert wird.
25      * @return Element der Grundmenge, das durch die
26      *   Struktur repraesentiert wird.
27      */
28     public Object getAtom() {
29         return _atom;
30     } // public Object getAtom() ...
31 } // class StructureSingleton extends Structure implements IStructureSingleton ...
```

Listing 130: Klasse *at.techmath.boltzmann.sampler.StructureSizeExceededException*

```
1  package at.techmath.boltzmann.sampler;
2
3  /**
4   * Exception, die geworfen wird, wenn die maximal erlaubte
5   * Groesse fuer Strukturen ueberschritten wurde.
6   * @author Stefan Schnabl (e0226245)
7   */
8  public class StructureSizeExceededException extends Exception {
9      private static final long serialVersionUID = -2141624026900298631L;
10     public StructureSizeExceededException() { super(); }
11     public StructureSizeExceededException(String message) { super(message); }
12     public StructureSizeExceededException(String message, Throwable cause) { super(message,
           cause); }
13     public StructureSizeExceededException(Throwable cause) { super(cause); }
14 } // public class StructureSizeExceededException extends Exception ...
```

Listing 131: Klasse *at.techmath.boltzmann.sampler.StructureSum*

```
1  package at.techmath.boltzmann.sampler;
2
3  /**
4   * Standard-Implementierung des IStructureSum-Interfaces.
5   * @author Stefan Schnabl (e0226245)
6   */
7  public class StructureSum extends Structure implements IStructureSum {
8      /**
9       * Erzeugt eine neue Instanz.
10      * @param structure Struktur, die vom Summen-Sampler gesampelt wurde.
11      * @throws IllegalArgumentException structure ist null.
12      */
13     public StructureSum(Structure structure) {
14         super();
15         if (structure == null) {
16             throw new IllegalArgumentException("structure = null.");
17         } // if (structure == null) ...
18         _structure = structure;
19     } // public StructureSum(Structure structure) ...
20
21     /**
22      * Struktur, die vom Summen-Sampler gesampelt wurde.
23      */
24     private Structure _structure;
25
26     /**
27      * Gibt die Struktur, die vom Summen-Sampler gesampelt wurde, zurueck.
28      * @return Struktur, die vom Summen-Sampler gesampelt wurde.
29      */
30     public IStructure getStructure() {
31         return _structure;
32     } // public IStructure getStructure() ...
33 } // public class StructureSum extends Structure implements IStructureSum ...
```

A.1.7 Package at.techmath.boltzmann.specification

Listing 132: Interface *at.techmath.boltzmann.specification.IIntegerLabel*

```
1  package at.techmath.boltzmann.specification;
2
3  /**
4   * Interface, das ein Label mit ganzzahligem Inhalt repraesentiert.
5   * @author Stefan Schnabl (e0226245)
6   */
7  public interface IIntegerLabel extends ILabel {
8      /**
9       * Gibt die Nummer, die das Label repraesentiert, zurueck.
10      * @return Nummer, die das Label repraesentiert.
11      */
12     long getNumber();
13 } // public interface IIntegerLabel extends ILabel ...
```

Listing 133: Interface *at.techmath.boltzmann.specification.ILabel*

```
1  package at.techmath.boltzmann.specification;
2
3  /**
4   * Interface, das ein Label, das in einer kombinatorischen
5   * Spezifikation verwendet wird, repraesentiert.
6   * @author Stefan Schnabl (e0226245)
7   */
8  public interface ILabel {
9      /**
10      * Gibt die Label-Collection zurueck, der das
11      * Label angehoert.
12      * @return Label-Collection, der das Label angehoert.
13      */
14     ILabelCollection getLabelCollection();
15
16     /**
17      * Gibt den Namen des Labels zurueck.
18      * @return Name des Labels.
19      */
20     String getName();
21
22     /**
23      * Gibt den Index des Labels innerhalb der LabelCollection zrueck.
24      * @return Index des Labels innerhalb der LabelCollection.
25      */
26     int getIndex();
27 } // public interface ILabel ...
```

Listing 134: Interface *at.techmath.boltzmann.specification.ILabelCollection*

```
1  package at.techmath.boltzmann.specification;
2
3  /**
4   * Sammlung der Labels, die fuer eine
5   * kombinatorische Spezifikation zur Verfuegung stehen.
6   * @author Stefan Schnabl (e0226245)
7   */
8  public interface ILabelCollection {
9      /**
10      * Gibt die Spezifikation zurueck, der die
11      * Label-Kollektion angehoert.
12      * @return Spezifikation, der die Label-Kollektion angehoert.
13      */
14     ISpecification getSpecification();
15
16     /**
17      * Gibt die Anzahl der Labels zurueck.
18      * @return Anzahl der Labels.
19      */
20     int getCount();
21
22     /**
23      * Gibt das Label mit dem angegebenen Index zurueck.
24      * Der Index muss zwischen 0 und getCount() - 1 liegen.
25      * @param index Index des gewuenschten Labels.
26      * @return Label mit dem angegebenen Index.
27      * @throws IndexOutOfBoundsException Wird geworfen,
28      *   wenn der Index ausserhalb des gueltigen Bereichs ist.
29      */
30     ILabel getLabelIndex(int index);
31
32     /**
33      * Gibt das Label mit dem angegebenen Namen zurueck.
34      * @param name Name des gewuenschten Labels.
35      * @return Label mit dem angegebenen Namen, falls vorhanden,
36      *   ansonsten null.
37      */
38     ILabel getLabel(String name);
39 } // public interface ILabelCollection ...
```

Listing 135: Interface *at.techmath.boltzmann.specification.INamedSpecies*

```
1  package at.techmath.boltzmann.specification;
2
3  /**
4   * Interface, das eine Spezies repraesentiert,
5   * die in der Spezifikation frei definierbar ist
6   * und einen Namen hat.
7   * @author Stefan Schnabl (e0226245)
8   */
9  public interface INamedSpecies extends ISpecies {
10     /**
11      * Gibt den Index der Spezies innerhalb der Spezifikation zurueck.
12      * @return Index der Spezies innerhalb der Spezifikation.
13      */
14     int getIndex();
15
16     /**
17      * Gibt den Namen der Spezies zurueck.
18      * @return Name der Spezies.
19      */
20     String getName();
21 } // public interface INamedSpecies extends ISpecies ...
```

Listing 136: Klasse *at.techmath.boltzmann.specification.IntegerLabel*

```
1  package at.techmath.boltzmann.specification;
2
3  import org.w3c.dom.Node;
4  import org.w3c.dom.NodeList;
5
6  /**
7   * Standard-Implementierung des IIntegerLabel-Interfaces.
8   * @author Stefan Schnabl (e0226245)
9   */
10 class IntegerLabel extends Label implements IIntegerLabel {
11     /**
12      * Name des Xml-Elements.
13      */
14     public static final String XML_ELEMENT_NAME = "integerlabel";
15
16     /**
17      * Erzeugt eine neue Instanz.
18      * @param name Name des Labels.
19      * @param number Nummer, die das Label repraesentiert.
20      */
21     public IntegerLabel(String name, long number) {
22         super(name);
23         _number = number;
24     } // public IntegerLabel(String name, long number) ...
25
26     /**
27      * Nummer, die das Label repraesentiert.
28      */
29     private long _number = 0L;
30
31     /**
32      * Gibt die Nummer, die das Label repraesentiert, zurueck.
33      * @return Nummer, die das Label repraesentiert.
34      */
35     public long getNumber() {
36         return _number;
37     } // public long getNumber() ...
38
39     /**
40      * Erzeugt aus dem gegebenen Xml-Element eine IntegerLabel-Instanz.
41      * @param name Name des Labels.
42      * @param xmlElement Xml-Element, aus dem das Label erzeugt werden soll.
43      * @return IntegerLabel-Instanz, die aus dem uebergebenen Xml-Element erzeugt wurde.
44      */
45     protected static IntegerLabel createFromXml(String name, org.w3c.dom.Element xmlElement)
46       throws SpecificationXmlLoadException {
47         if (xmlElement == null) throw new IllegalArgumentException("xmlElement = null.");
48         // Repraesentierte Nummer ermitteln
49         long numberContent = 0L;
50         NodeList childs = xmlElement.getChildNodes();
51         if (childs != null) {
52             StringBuilder sb = new StringBuilder();
53             int length = childs.getLength();
54             for (int c = 0; c < length; c++) {
55                 Node currentNode = childs.item(c);
56                 if (currentNode != null && currentNode.getNodeType() == Node.TEXT_NODE) {
57                     String currentText = currentNode.getNodeValue();
58                     if (currentText != null) {
59                         sb.append(currentText);
60                     } // if (currentText != null) ...
61                 } // if (currentNode != null && currentNode.getNodeType() == Node.TEXT_NODE)
                       ...
62             } // for (int c = 0; c < length; c++) ...
63             try {
64                 numberContent = Long.parseLong(sb.toString().trim(), 10);
65             } catch (NumberFormatException numberFormatException) {
66                 throw new SpecificationXmlLoadException("Ungueltige Zahl im angegebenen Integer
                       -Label.", numberFormatException);
67             }
68             return new IntegerLabel(name, numberContent);
69         } // if (childs != null) ....
70         return null;
71     } // protected static IntegerLabel createFromXml(String name, org.w3c.dom.Element
           xmlElement) ...
72
73     /**
74      * Gibt die String-Repraesentation der Instanz zurueck.
75      * @return String-Repraesentation der Instanz.
76      */
```

```
77      @Override
78      public String toString() {
79          StringBuilder sb = new StringBuilder();
80          sb.append("Integer-Label (\"");
81          sb.append(getName());
82          sb.append("\", ");
83          sb.append(getNumber());
84          sb.append(")");
85          return sb.toString();
86      } // public String toString() ...
87  } // class IntegerLabel extends Label implements IIntegerLabel ...
```

Listing 137: Interface *at.techmath.boltzmann.specification.ISpecies*

```
1  package at.techmath.boltzmann.specification;
2
3  /**
4   * Interface, das eine kombinatorische Spezies repraesentiert.
5   * @author Stefan Schnabl (e0226245)
6   */
7  public interface ISpecies {
8      /**
9       * Gibt die Spezifikation, der die Spezies
10      * angehoert, zurueck.
11      * @return Spezifikation der Spezies, der die Spezies angehoert.
12      */
13     ISpecification getSpecification();
14 } // public interface ISpecies ...
```

Listing 138: Interface *at.techmath.boltzmann.specification.ISpeciesCycle*

```
1  package at.techmath.boltzmann.specification;
2
3  /**
4   * Interface, das die Spezies der Zyklen repraesentiert.
5   * @author Stefan Schnabl (e0226245)
6   */
7  public interface ISpeciesCycle extends INamedSpecies {
8      /**
9       * Gibt eine Referenz auf die Spezies zurueck,
10      * von der Zyklen gebildet werden sollen.
11      * @return Referenz auf die Spezies, von
12      *   der Zyklen gebildet werden sollen.
13      */
14     ISpeciesReference getSpeciesParameter();
15 } // public interface ISpeciesCycle extends INamedSpecies ...
```

Listing 139: Interface *at.techmath.boltzmann.specification.ISpeciesOne*

```
1 package at.techmath.boltzmann.specification;
2
3 /**
4  * Interface, das die Spezies der Charakteristik der leeren Menge repraesentiert.
5  * @author Stefan Schnabl (e0226245)
6  */
7 public interface ISpeciesOne extends ISpecies {
8 } // public interface ISpeciesOne extends ISpecies ...
```

Listing 140: Interface *at.techmath.boltzmann.specification.ISpeciesProduct*

```
1  package at.techmath.boltzmann.specification;
2
3  /**
4   * Interface, das die Produkt-Spezies repraesentiert.
5   * @author Stefan Schnabl (e0226245)
6   */
7  public interface ISpeciesProduct extends INamedSpecies {
8      /**
9       * Gibt die Anzahl der Faktoren zurueck.
10      * @return Anzahl der Faktoren.
11      */
12     int getFactorCount();
13
14     /**
15      * Gibt den Faktor mit dem angegebenen Index zurueck.
16      * Der Index muss zwischen 0 und getFactorCount() - 1 liegen.
17      * @param index Index des gewuenschten Faktoren.
18      * @return Faktor mit dem angegebenen Index.
19      * @throws IndexOutOfBoundsException Der angegebene Index
20      *   liegt ausserhalb des gueltigen Bereichs.
21      */
22     ISpeciesReference getFactorIndex(int index);
23 } // public interface ISpeciesProduct extends INamedSpecies ...
```

Listing 141: Interface *at.techmath.boltzmann.specification.ISpeciesReference*

```
1  package at.techmath.boltzmann.specification;
2
3  /**
4   * Interface, das einen Verweis auf eine Spezies darstellt.
5   * @author Stefan Schnabl (e0226245)
6   */
7  public interface ISpeciesReference {
8      /**
9       * Gibt die Spezies zurueck, von der die
10      * Referenz verwendet wird.
11      * @return Spezies, von der die Referenz verwendet wird.
12      */
13     ISpecies getParentSpecies();
14
15     /**
16      * Gibt das Label, das der Referenz zugeordnet ist,
17      * zurueck.
18      * @return Label, das der Referenz zugeordnet ist,
19      *   falls vorhanden, ansonsten null.
20      */
21     ILabel getLabel();
22
23     /**
24      * Gibt die Spezies zurueck, die referenziert wird.
25      * @return Spezies, die referenziert wird.
26      */
27     ISpecies getSpecies();
28 } // public interface ISpeciesReference ...
```

Listing 142: Interface *at.techmath.boltzmann.specification.ISpeciesSequence*

```
1  package at.techmath.boltzmann.specification;
2
3  /**
4   * Interface, das die Spezies der endlichen Folgen repraesentiert.
5   * @author Stefan Schnabl (e0226245)
6   */
7  public interface ISpeciesSequence extends INamedSpecies {
8      /**
9       * Gibt eine Referenz auf die Spezies zurueck,
10      * von der endliche Folgen gebildet werden sollen.
11      * @return Referenz auf die Spezies, von der
12      *   endliche Folgen gebildet werden sollen.
13      */
14     ISpeciesReference getSpeciesParameter();
15 } // public interface ISpeciesSequence extends INamedSpecies ...
```

Listing 143: Interface *at.techmath.boltzmann.specification.ISpeciesSet*

```
1  package at.techmath.boltzmann.specification;
2
3  /**
4   * Interface, das die Spezies der Mengen repraesentiert.
5   * @author Stefan Schnabl (e0226245)
6   */
7  public interface ISpeciesSet extends INamedSpecies {
8      /**
9       * Gibt eine Referenz auf die Spezies zurueck,
10      * von der Mengen gebildet werden sollen.
11      * @return Referenz auf die Spezies, von der
12      *   Mengen gebildet werden sollen.
13      */
14     ISpeciesReference getSpeciesParameter();
15 } // public interface ISpeciesSet extends INamedSpecies ...
```

Listing 144: Interface *at.techmath.boltzmann.specification.ISpeciesSingleton*

```
1 package at.techmath.boltzmann.specification;
2
3 /**
4  * Interface, das die Singleton-Spezies repraesentiert.
5  * @author Stefan Schnabl (e0226245)
6  */
7 public interface ISpeciesSingleton extends ISpecies {
8 } // public interface ISpeciesSingleton extends ISpecies ...
```

Listing 145: Interface *at.techmath.boltzmann.specification.ISpeciesSum*

```
1  package at.techmath.boltzmann.specification;
2
3  /**
4   * Interface, das die Summen-Spezies repraesentiert.
5   * @author Stefan Schnabl (e0226245)
6   */
7  public interface ISpeciesSum extends INamedSpecies {
8      /**
9       * Gibt die Anzahl der Summanden zurueck.
10      * @return Anzahl der Summanden.
11      */
12     int getAddendCount();
13
14     /**
15      * Gibt den Summanden mit dem angegebenen Index zurueck.
16      * Der Index muss zwischen 0 und getAddendCount() - 1 liegen.
17      * @param index Index des gewuenschten Summanden.
18      * @return Summand mit dem angegebenen Index.
19      * @throws IndexOutOfBoundsException Der angegebene Index
20      *   liegt ausserhalb des gueltigen Bereichs.
21      */
22     ISpeciesReference getAddendIndex(int index);
23 } // public interface ISpeciesSum extends INamedSpecies ...
```

Listing 146: Interface *at.techmath.boltzmann.specification.ISpecification*

```
1  package at.techmath.boltzmann.specification;
2
3  /**
4   * Interface, das eine kombinatorische Spezifikation repraesentiert.
5   * @author Stefan Schnabl (e0226245)
6   */
7  public interface ISpecification {
8      /**
9       * Gibt die Spezifikations-Kollektion zurueck,
10      * der die Spezifikation angehoert.
11      * @return Spezifikations-Kollektion, der die Spezifikation angehoert.
12      */
13     ISpecificationCollection getSpecificationCollection();
14
15     /**
16      * Gibt die Anzahl der Spezies, aus denen sich
17      * das kombinatorische System zusammensetzt,
18      * zurueck.
19      * @return Anzahl der Spezies, aus denen sich
20      *   das kombinatorisch System zusammensetzt.
21      */
22     int getCount();
23
24     /**
25      * Gibt den Namen der kominatorischen Spezifikation zurueck.
26      * @return Name der kombinatorischen Spezifikation.
27      */
28     String getName();
29
30     /**
31      * Gibt die Spezies mit dem angegebenen Index
32      * zurueck. Der Index muss im Bereich 0 .. getCount() - 1 liegen.
33      * @param index Index der gewuenschten Spezies.
34      * @return Spezies mit dem angegebenen Index.
35      * @throws IndexOutOfBoundsException Wird geworfen,
36      *   wenn der angegebene Index ausserhalb des gueltigen
37      *   Bereichs ist.
38      */
39     INamedSpecies getSpeciesIndex(int index);
40
41     /**
42      * Gibt die Spezies mit dem angegebenen Namen
43      * zurueck.
44      * @param name Name der Spezies, die zurueckgegeben werden soll.
45      * @return Spezies mit dem angegebenen Namen, falls vorhanden,
46      *   ansonsten null.
47      */
48     INamedSpecies getSpecies(String name);
49
50     /**
51      * Gibt die Singleton-Spezies zurueck.
52      * @return Singleton-Spezies.
53      */
54     ISpeciesSingleton getSpeciesSingleton();
55
56     /**
57      * Gibt die Spezies der Charakteristik der leeren Menge zurueck.
58      * @return Spezies der Charakteristik der leeren Menge.
59      */
60     ISpeciesOne getSpeciesOne();
61
62     /**
63      * Gibt die Standard-Spezies zurueck.
64      * @return Standard-Spezies.
65      */
66     INamedSpecies getDefaultSpecies();
67
68     /**
69      * Gibt die Label-Collection der Labels, die von
70      * der Spezifikation verwendet werden, zurueck.
71      * @return Label-Collection der Labels, die
72      *   von der Spezifikation verwendet werden.
73      */
74     ILabelCollection getLabelCollection();
75 } // public interface ISpecification
```

Listing 147: Interface *at.techmath.boltzmann.specification.ISpecificationCollection*

```
1  package at.techmath.boltzmann.specification;
2
3  /**
4   * Interface, das eine Sammlung von kombinatorischen Spezifikationen
5   * repraesentiert.
6   * @author Stefan Schnabl (e0226245)
7   */
8  public interface ISpecificationCollection {
9      /**
10      * Gibt die Anzahl der kombinatorischen Spezifikationen
11      * zurueck.
12      * @return Anzahl der kombinatorischen Spezifikationen.
13      */
14     int getCount();
15
16     /**
17      * Gibt die Spezifikation mit dem angegebenen Index
18      * zurueck. Der Index muss zwischen 0 und getCount() - 1 liegen.
19      * @param index Index der Spezifikation, die ermittelt werden soll.
20      * @return Spezifikation mit dem angegebenen Index.
21      * @throws IllegalArgumentException Der angegebene Index liegt ausserhalb
22      *   des gueltigen Bereichs.
23      */
24     ISpecification getSpecificationIndex(int index);
25
26     /**
27      * Gibt die Spezifikation mit dem angegebenen Namen zurueck.
28      * @param name Name der gewuenschten Spezifikation.
29      * @return Spezifikation mit dem angegebenen Namen,
30      *   falls vorhanden, ansonsten null.
31      */
32     ISpecification getSpecification(String name);
33
34     /**
35      * Gibt die Standard-Spezifikation zurueck.
36      * @return Standard-Spezifikation.
37      */
38     ISpecification getDefaultSpecification();
39 } // public interface ISpecificationCollection ...
```

Listing 148: Interface *at.techmath.boltzmann.specification.IStringLabel*

```
1  package at.techmath.boltzmann.specification;
2
3  /**
4   * Interface, das ein Label mit Text-Inhalt repraesentiert.
5   * @author Stefan Schnabl (e0226245)
6   */
7  public interface IStringLabel extends ILabel {
8      /**
9       * Gibt den Textinhalt des Labels zurueck.
10      * @return Textinhalt des Labels.
11      */
12     String getText();
13 } // public interface IStringLabel extends ILabel ...
```

Listing 149: Klasse *at.techmath.boltzmann.specification.Label*

```
1  package at.techmath.boltzmann.specification;
2
3  import org.w3c.dom.NamedNodeMap;
4  import org.w3c.dom.Node;
5
6  /**
7   * Basis-Implementierung des ILabel-Interfaces.
8   * @author Stefan Schnabl (e0226245)
9   */
10 abstract class Label implements ILabel {
11     /**
12      * Erzeugt eine neue Instanz.
13      * @param name Name des Labels.
14      */
15     public Label (String name) {
16         super();
17         if (name == null) throw new IllegalArgumentException("name = null.");
18         name = name.trim();
19         if (name.length() <= 0) throw new IllegalArgumentException("Es muss ein Name mit einer
               positiven Laenge angegeben werden.");
20         _name = name;
21         _normName = name.toLowerCase();
22     } // public Label (String name) ...
23
24     /**
25      * Index des Labels.
26      */
27     private int _index = -1;
28
29     /**
30      * Setzt den Index des Labels.
31      * @param index Index des Labels.
32      */
33     protected void setIndex(int index) {
34         _index = index;
35     } // protected void setIndex(int index) ...
36
37     /**
38      * Gibt den Index des Labels innerhalb der LabelCollection zrueck.
39      * @return Index des Labels innerhalb der LabelCollection.
40      */
41     public int getIndex() {
42         return _index;
43     } // public int getIndex() ...
44
45     /**
46      * LabelCollection, der das Label angehoert.
47      */
48     private LabelCollection _parent = null;
49
50     /**
51      * Name des Labels.
52      */
53     private String _name = null;
54
55     /**
56      * Normierter Name des Labels.
57      */
58     private String _normName = null;
59
60     /**
61      * Setzt die LabelCollection, der das
62      * Label angehoert.
63      * @param parent LabelCollection, der das Label angehoert.
64      */
65     protected void setParent (LabelCollection parent) {
66         _parent = parent;
67     } // protected void setParent (LabelCollection parent) ...
68
69     /**
70      * Gibt die Label-Collection zurueck, der das
71      * Label angehoert.
72      * @return Label-Collection, der das Label angehoert.
73      */
74     public ILabelCollection getLabelCollection() {
75         return _parent;
76     } // public ILabelCollection getLabelCollection() ...
77
78     /**
```

```
79      * Gibt den Namen des Labels zurueck.
80      * @return Name des Labels.
81      */
82     public String getName() {
83         return _name;
84     } // public String getName() ...
85
86     /**
87      * Gibt den normierten Namen des Labels zurueck.
88      * @return Normierter Name des Labels.
89      */
90     public String getNormName() {
91         return _normName;
92     } // public String getNormName() ...
93
94     /**
95      * Erzeugt ein Label, das durch das angegebene
96      * Xml-Element spezifiziert ist.
97      * @param xmlElement Xml-Element, aus dem das Label erzeugt werden soll.
98      * @return XmlElement
99      */
100    public static Label createFromXml (org.w3c.dom.Element xmlElement)
101      throws SpecificationXmlLoadException {
102        if (xmlElement == null) throw new IllegalArgumentException("xmlElement = null.");
103        String nodeName = xmlElement.getLocalName();
104        if (nodeName == null) throw new IllegalArgumentException("xmlElement hat keinen Namen."
               );
105        // Uebergebenen Namen des Labels bestimmen.
106        String labelName = null;
107        {
108            NamedNodeMap attributes = xmlElement.getAttributes();
109            if (attributes != null) {
110                int length = attributes.getLength();
111                for (int currentIndex = 0; currentIndex < length; currentIndex++) {
112                    Node currentAttribute = attributes.item(currentIndex);
113                    if (currentAttribute != null) {
114                        String namespaceURI = currentAttribute.getNamespaceURI();
115                        if (namespaceURI == null || SpecificationCollection.
                               SPECIFICATION_NAMESPACE.equals(namespaceURI)) {
116                            String normAttributeName = currentAttribute.getLocalName();
117                            if (normAttributeName != null) {
118                                normAttributeName = normAttributeName.trim().toLowerCase();
119                                if (normAttributeName.equals("name")) {
120                                    labelName = currentAttribute.getNodeValue();
121                                    if (labelName != null) {
122                                        labelName = labelName.trim ();
123                                    } // if (labelName != null) ...
124                                    break;
125                                } // if (normAttributeName.equals("name")) ...
126                            } // if (normAttributeName != null) ...
127                        } // if (namespaceURI == null || SpecificationCollection.
                               SPECIFICATION_NAMESPACE.equals(namespaceURI)) ...
128                    } // if (currentAttribute != null) ...
129                } // for (int currentIndex = 0; currentIndex < length; currentIndex++) ...
130            } // if (attributes != null) ...
131            if (labelName == null || labelName.length() <= 0) {
132                throw new SpecificationXmlLoadException("Label-Element ohne Namen gefunden.");
133            }
134        }
135        nodeName = nodeName.trim().toLowerCase();
136        if (nodeName.equals(StringLabel.XML_ELEMENT_NAME)) {
137            return StringLabel.createFromXml(labelName, xmlElement);
138        } else if (nodeName.equals(IntegerLabel.XML_ELEMENT_NAME)) {
139            return IntegerLabel.createFromXml(labelName, xmlElement);
140        } else {
141            throw new SpecificationXmlLoadException("Unbekanntes Label-Element");
142        }
143    } // public static Label createFromXml (org.w3c.dom.Element xmlElement) ...
144 } // abstract class Label implements ILabel ...
```

Listing 150: Klasse *at.techmath.boltzmann.specification.LabelCollection*

```
1  package at.techmath.boltzmann.specification;
2
3  import java.util.Vector;
4  import org.w3c.dom.Node;
5  import org.w3c.dom.NodeList;
6  import org.w3c.dom.Element;
7
8  /**
9   * Standard-Implementierung des ILabelCollection-Interfaces.
10  * @author Stefan Schnabl (e0226245)
11  */
12 class LabelCollection implements ILabelCollection {
13     /**
14      * Name des Xml-Elements.
15      */
16     public static final String XML_ELEMENT_NAME = "labels";
17
18     /**
19      * Spezifikation, der die LabelCollection angehoert.
20      */
21     private Specification _parent = null;
22
23     /**
24      * Gibt die Spezifikation zurueck, der die
25      * Label-Kollektion angehoert.
26      * @return Spezifikation, der die Label-Kollektion angehoert.
27      */
28     public ISpecification getSpecification() {
29         return _parent;
30     } // public ISpecification getSpecification() ...
31
32     /**
33      * Setzt die Spezifikation, der die
34      * Label-Kollektion angehoert.
35      * @param parent Spezifikation, der die Label-Kollektion angehoert.
36      */
37     protected void setParent(Specification parent) {
38         _parent = parent;
39     } // protected void setParent(Specification parent) ...
40
41     /**
42      * Liste der Labels, die der Collection angehoeren.
43      */
44     private Vector<Label> _labelList = new Vector<Label>();
45
46     /**
47      * Gibt die Anzahl der Labels zurueck.
48      * @return Anzahl der Labels.
49      */
50     public int getCount() {
51         return _labelList.size();
52     } // public int getCount() ...
53
54     /**
55      * Gibt das Label mit dem angegebenen Index zurueck.
56      * Der Index muss zwischen 0 und getCount() - 1 liegen.
57      * @param index Index des gewuenschten Labels.
58      * @return Label mit dem angegebenen Index.
59      * @throws IndexOutOfBoundsException Wird geworfen,
60      *   wenn der Index ausserhalb des gueltigen Bereichs ist.
61      */
62     public ILabel getLabelIndex(int index) {
63         if (index < 0 || index >= _labelList.size()) throw new IndexOutOfBoundsException();
64         return _labelList.elementAt(index);
65     } // public ILabel getLabelIndex(int index) ...
66
67     /**
68      * Gibt das Label mit dem angegebenen Namen zurueck.
69      * @param name Name des gewuenschten Labels.
70      * @return Label mit dem angegebenen Namen, falls vorhanden,
71      *   ansonsten null.
72      */
73     public ILabel getLabel(String name) {
74         if (name != null) {
75             String normName = name.trim().toLowerCase();
76             int l = 0;
77             int r = _labelList.size() - 1;
78             while (l <= r) {
79                 int m = (l + r) / 2;
```

```
80                  Label currentLabel = _labelList.elementAt(m);
81                  String currentName = currentLabel.getNormName();
82                  int comparison = currentName.compareTo(normName);
83                  if (comparison < 0) {
84                      l = m + 1;
85                  } else if (comparison > 0) {
86                      r = m - 1;
87                  } else {
88                      return currentLabel;
89                  }
90              } // while (l <= r) ...
91          } // if (name != null) ...
92          return null;
93      } // public ILabel getLabel(String name) ...
94
95      /**
96       * Fuegt das angegebene Label zur Collection hinzu.
97       * Der Parent des Labels wird auf die aktuelle Instanz gesetzt.
98       * @param label Label, das hinzugefuegt werden soll.
99       * @return true, falls das Label hinzugefuegt werden konnte,
100      *   da noch kein zweites Label mit dem selben Namen existiert,
101      *   ansonsten false.
102      * @throws IllegalArgumentException Wird geworfen, wenn das
103      *   angegebene Label null ist.
104      */
105     private boolean addLabel(Label label) {
106         if (label == null) throw new IllegalArgumentException("label = null.");
107         int l = 0;
108         int r = _labelList.size() - 1;
109         String newName = label.getNormName();
110         while (l <= r) {
111             int m = (l + r) / 2;
112             Label currentLabel = _labelList.elementAt(m);
113             String currentName = currentLabel.getNormName();
114             int comparison = currentName.compareTo(newName);
115             if (comparison < 0) {
116                 l = m + 1;
117             } else if (comparison > 0) {
118                 r = m - 1;
119             } else { // if (comparison == 0) ...
120                 // Es gibt schon ein Label mit dem Namen.
121                 return false;
122             } // if (comparison == 0) ...
123         } // while (l <= r) ...
124         _labelList.insertElementAt(label, l);
125         label.setParent(this);
126         return true;
127     } // private boolean addLabel(Label label) ...
128
129     /**
130      * Erzeugt aus dem uebergebenen Xml-Element eine LabelCollection.
131      * @param xmlElement Xml-Element, aus dem die LabelCollection erzeugt werden soll.
132      * @return LabelCollection, die aus dem uebergebenen Xml-Element erzeugt wurde.
133      * @throws at.techmath.boltzmann.specification.SpecificationXmlLoadException Wird geworfen,
134      *   wenn die Struktur des Xml-Dokuments ungueltig ist.
135      * @throws IllegalArgumentException Wird geworfen, wenn das
136      *   angegebene Xml-Element nil ist.
137      */
138     public static LabelCollection createFromXml(org.w3c.dom.Element xmlElement)
139       throws SpecificationXmlLoadException {
140         if (xmlElement == null) throw new IllegalArgumentException("xmlElement = null.");
141         LabelCollection result = new LabelCollection();
142         NodeList childNodes = xmlElement.getChildNodes();
143         if (childNodes != null) {
144             int length = childNodes.getLength();
145             for (int c = 0; c < length; c++) {
146                 Node currentNode = childNodes.item(c);
147                 if (currentNode != null && currentNode.getNodeType() == Node.ELEMENT_NODE &&
                        currentNode instanceof Element) {
148                     Element currentElement = (Element) currentNode;
149                     if (SpecificationCollection.SPECIFICATION_NAMESPACE.equals(currentElement.
                            getNamespaceURI())) {
150                         Label newLabel = Label.createFromXml((Element) currentNode);
151                         if (!result.addLabel(newLabel)) {
152                             throw new SpecificationXmlLoadException("Mehrere Label mit dem
                                    selben Namen angegeben.");
153                         } // if (!result.addLabel(newLabel)) ...
154                     } //  if (SpecificationCollection.SPECIFICATION_NAMESPACE.equals(
                            currentElement.getNamespaceURI())) ...
```

```
155                 } // if (currentNode != null && currentNode.getNodeType() == Node.ELEMENT_NODE)
                        ...
156             } // for (int c = 0; c < length; c++) ...
157         } // if (childNodes != null) ...
158         // Index der Labels setzen.
159         int size = result._labelList.size();
160         for (int c = 0; c < size; c++) {
161             Label currentLabel = result._labelList.elementAt(c);
162             currentLabel.setIndex(c);
163         } // for (int c = 0; c < size; c++) ...
164         return result;
165     } // public static LabelCollection createFromXml(org.w3c.dom.Element xmlElement) ...
166 
167     /**
168      * Gibt eine String-Repraesentation der Instanz zurueck.
169      * @return String-Repraesentation der Instanz.
170      */
171     @Override
172     public String toString() {
173         StringBuilder sb = new StringBuilder();
174         sb.append("    Label-Collection\n");
175         int count = getCount();
176         for (int c = 0; c < count; c++) {
177             ILabel currentLabel = getLabelIndex(c);
178             if (currentLabel != null) {
179                 sb.append("      ");
180                 sb.append(currentLabel.toString());
181                 sb.append("\n");
182             } // if (currentLabel != null) ...
183         } // for (int c = 0; c < count; c++) ...
184         return sb.toString();
185     } // public String toString() ...
186 } // class LabelCollection implements ILabelCollection ...
```

Listing 151: Klasse *at.techmath.boltzmann.specification.NamedSpecies*

```
1  package at.techmath.boltzmann.specification;
2
3  import org.w3c.dom.Node;
4  import org.w3c.dom.NamedNodeMap;
5
6  /**
7   * Abstrakte Basisklasse fuer Spezies,
8   * die das INamedSpecies-Interface implementieren.
9   * @author Stefan Schnabl (e0226245)
10  */
11 abstract class NamedSpecies extends Species implements INamedSpecies {
12     /**
13      * Erzeugt eine neue Instanz.
14      * @param name Name der Spezies.
15      * @throws IllegalArgumentException Wird geworfen, wenn
16      *   der Name null oder leer ist.
17      */
18     public NamedSpecies(String name) {
19         super();
20         if (name == null) throw new IllegalArgumentException("name = null.");
21         name = name.trim();
22         if (name.length() <= 0) throw new IllegalArgumentException("Es muss ein Name mit einer
               positiven Laenge angegeben werden.");
23         _name = name;
24         _normName = name.toLowerCase();
25     } // public NamedSpecies(String name) ...
26
27     /**
28      * Index der Spezies innerhalb der Spezifikation.
29      */
30     private int _index = -1;
31
32     /**
33      * Gibt den Index der Spezies innerhalb der Spezifikation zurueck.
34      * @return Index der Spezies innerhalb der Spezifikation.
35      */
36     public int getIndex() {
37         return _index;
38     } // public int getIndex() ...
39
40     /**
41      * Setzt den Index der Spezies innerhalb der Spezifikation.
42      * @param index Index der Spezies innerhalb der Spezifikation.
43      */
44     protected void setIndex(int index) {
45         _index = index;
46     } // protected void setIndex(int index) ...
47
48     /**
49      * Name der Spezies.
50      */
51     private String _name;
52
53     /**
54      * Normierter Name der Spezies.
55      */
56     private String _normName;
57
58     /**
59      * Gibt den Namen der Spezies zurueck.
60      * @return Name der Spezies.
61      */
62     public String getName() {
63         return _name;
64     } // public String getName() ...
65
66     /**
67      * Gibt den normierten Namen der Spezies zurueck.
68      * @return Normierter Name der Spezies.
69      */
70     protected String getNormName() {
71         return _normName;
72     } // protected String getNormName() ...
73
74     /**
75      * Erzeugt aus dem angegebenen Xml-Element die entsprechende Spezies.
76      * @param xmlElement Xml-Element, aus dem die Spezies erzeugt werden soll.
77      * @return Spezies, die anhand des angegebenen Xml-Elements erzeugt wurde.
78      * @throws at.techmath.boltzmann.specification.SpecificationXmlLoadException Wird geworfen,
```

```
79       *   wenn das angegebene Xml-Element nicht der erwarteten Struktur entspricht.
80       * @throws IllegalArgumentException Wird geworfen, wenn das angegebene
81       *   Xml-Element null ist bzw. nicht vom erwarteten Typ ist.
82       */
83      public static NamedSpecies createFromXml(org.w3c.dom.Element xmlElement)
84        throws SpecificationXmlLoadException {
85          if (xmlElement == null) throw new IllegalArgumentException("xmlElement = null.");
86          String elementName = xmlElement.getLocalName();
87          if (elementName == null) { elementName = ""; } else { elementName = elementName.trim().
                toLowerCase(); }
88          if (!isKnownNamedSpeciesElement(elementName) ||
89              !SpecificationCollection.SPECIFICATION_NAMESPACE.equals(xmlElement.getNamespaceURI
                    ())) {
90              throw new IllegalArgumentException("Spezies-Element erwartet.");
91          }
92
93          // Namen der Spezies laden
94          String nameSpezies = null;
95          {
96              NamedNodeMap attributes = xmlElement.getAttributes();
97              if (attributes != null) {
98                  int length = attributes.getLength();
99                  for (int c = 0; c < length; c++) {
100                     Node currentAttribute = attributes.item(c);
101                     if (currentAttribute != null) {
102                         String namespaceURI = currentAttribute.getNamespaceURI();
103                         if (namespaceURI == null || SpecificationCollection.
                                SPECIFICATION_NAMESPACE.equals(namespaceURI)) {
104                             String currentAttributeName = currentAttribute.getLocalName();
105                             if (currentAttributeName == null) { currentAttributeName = ""; }
                                    else { currentAttributeName = currentAttributeName.trim().
                                    toLowerCase(); }
106                             if (currentAttributeName.equals("name")) {
107                                 if (nameSpezies != null) {
108                                     throw new SpecificationXmlLoadException("Name der Spezies
                                            mehrmals angegeben.");
109                                 } // if (nameSpezies != null) ...
110                                 nameSpezies = currentAttribute.getNodeValue();
111                                 if (nameSpezies == null) {
112                                     nameSpezies = "";
113                                 } else {
114                                     nameSpezies = nameSpezies.trim();
115                                 }
116                             } // if (currentAttributeName.equals("name")) ...
117                         } // if (namespaceURI == null || SpecificationCollection.
                                SPECIFICATION_NAMESPACE.equals(namespaceURI)) ...
118                     } // if (currentAttribute != null) ...
119                 } // for (int c = 0; c < length; c++0 ...
120             } // if (attributes != null) ...
121         }
122         if (nameSpezies == null || nameSpezies.length() <= 0) {
123             throw new SpecificationXmlLoadException("Kein oder leerer Name fuer Spezies
                    angegeben.");
124         } // if (nameSpezies == null || nameSpezies.length() <= 0) ...
125
126         NamedSpecies result = null;
127         if (elementName.equals(SpeciesSet.XML_ELEMENT_NAME)) {
128             result = SpeciesSet.createFromXml(nameSpezies, xmlElement);
129         } else if (elementName.equals(SpeciesSequence.XML_ELEMENT_NAME)) {
130             result = SpeciesSequence.createFromXml(nameSpezies, xmlElement);
131         } else if (elementName.equals(SpeciesCycle.XML_ELEMENT_NAME)) {
132             result = SpeciesCycle.createFromXml(nameSpezies, xmlElement);
133         } else if (elementName.equals(SpeciesSum.XML_ELEMENT_NAME)) {
134             result = SpeciesSum.createFromXml(nameSpezies, xmlElement);
135         } else if (elementName.equals(SpeciesProduct.XML_ELEMENT_NAME)) {
136             result = SpeciesProduct.createFromXml(nameSpezies, xmlElement);
137         }
138         if (result == null) {
139             throw new SpecificationXmlLoadException("Unbekannte Spezies.");
140         }
141         return result;
142     } // public static NamedSpecies createFromXml(org.w3c.dom.Element xmlElement) ...
143
144     /**
145      * Ermittelt, ob es sich beim angegebenen Namen um
146      * einen bekannten Xml-Element-Namen einer Spezies handelt.
147      * @param name Name der geprueft werden soll.
148      * @return true, falls es sich um einen bekannten Spezies-Xml-Element-Namen handelt,
149      *   ansonsten false.
150      */
```

```
151     protected static boolean isKnownNamedSpeciesElement(String name) {
152         name = name == null ? "" : name.trim().toLowerCase();
153         return
154             SpeciesSet.XML_ELEMENT_NAME.equals(name) ||
155             SpeciesSequence.XML_ELEMENT_NAME.equals(name) ||
156             SpeciesCycle.XML_ELEMENT_NAME.equals(name) ||
157             SpeciesSum.XML_ELEMENT_NAME.equals(name) ||
158             SpeciesProduct.XML_ELEMENT_NAME.equals(name);
159     } // protected static boolean isKnownNamedSpeciesElement(String name) ...
160 
161     /**
162      * Beendet den Xml-Ladevorgang.
163      * @throws at.techmath.boltzmann.specification.SpecificationXmlLoadException Wird geworfen,
164      *   wenn ein unbekanntes Objekt referenziert wird.
165      */
166     protected abstract void finishXmlLoad() throws SpecificationXmlLoadException;
167 
168 } // abstract class NamedSpecies extends Species implements INamedSpecies ...
```

Listing 152: Klasse *at.techmath.boltzmann.specification.NamedSpeciesReference*

```
1  package at.techmath.boltzmann.specification;
2
3  import org.w3c.dom.Node;
4  import org.w3c.dom.NamedNodeMap;
5
6  /**
7   * Referenz auf eine benannte Spezies.
8   * @author Stefan Schnabl (e0226245)
9   */
10 class NamedSpeciesReference extends SpeciesReference {
11     /**
12      * Name des Xml-Elements.
13      */
14     protected static final String XML_ELEMENT_NAME = "ref";
15
16     /**
17      * Name der Spezies.
18      */
19     private String _speciesName = null;
20
21     /**
22      * Spezies, die referenziert wird.
23      */
24     private NamedSpecies _species = null;
25
26     /**
27      * Gibt die Spezies zurueck, die referenziert wird.
28      * @return Spezies, die referenziert wird.
29      */
30     public ISpecies getSpecies() {
31         return _species;
32     } // public ISpecies getSpecies() ...
33
34     /**
35      * Erzeugt aus dem uebergebenen Xml-Element eine
36      * NamedSpecificationReference-Instanz.
37      * @param xmlElement Xml-Element, aus dem eine NamedSpecificationReference-Instanz
38      *   erzeugt werden soll.
39      * @return NamedSpecificationReference-Instanz, die aus dem
40      *   ubergebenen Xml-Element erzeugt wurde.
41      * @throws IllegalArgumentException Wird geworfen, wenn das angegebene Xml-Element
42      *   null ist oder nicht vom erwarteten Typ ist.
43      * @throws at.techmath.boltzmann.specification.SpecificationXmlLoadException Wird geworfen,
44      *   wenn das Xml-Element nicht die erwartete Struktur besitzt.
45      */
46     public static NamedSpeciesReference createFromXml(org.w3c.dom.Element xmlElement)
47       throws SpecificationXmlLoadException {
48         if (xmlElement == null) throw new IllegalArgumentException("xmlElement = null.");
49         String elementName = xmlElement.getLocalName();
50         if (elementName == null) { elementName = ""; } else { elementName = elementName.trim().
               toLowerCase(); }
51         if (!XML_ELEMENT_NAME.equals(elementName) ||
52             !SpecificationCollection.SPECIFICATION_NAMESPACE.equals(xmlElement.getNamespaceURI
                   ())) {
53             throw new IllegalArgumentException("\"" + XML_ELEMENT_NAME + "\"-Element erwartet."
                   );
54         }
55         // Namen der Spezies ermitteln
56         String nameSpecies = null;
57         {
58             NamedNodeMap attributes = xmlElement.getAttributes();
59             if (attributes != null) {
60                 int length = attributes.getLength();
61                 for (int c = 0; c < length; c++) {
62                     Node currentAttribute = attributes.item(c);
63                     if (currentAttribute != null) {
64                         String namespaceURI = currentAttribute.getNamespaceURI();
65                         if (namespaceURI == null || SpecificationCollection.
                               SPECIFICATION_NAMESPACE.equals(namespaceURI)) {
66                             String currentAttributeName = currentAttribute.getLocalName();
67                             if (currentAttributeName == null) {
68                                 currentAttributeName = "";
69                             } else { // if (currentAttributeName != null) ...
70                                 currentAttributeName = currentAttributeName.trim().toLowerCase
                                       ();
71                             } // if (currentAttributeName != null) ...
72                             if (currentAttributeName.equals("name")) {
73                                 if (nameSpecies != null) {
```

```
74                                  throw new SpecificationXmlLoadException("Name der
                                        referenzierten Spezies mehrfach angegeben.");
75                              } // if (nameSpecies != null) ...
76                              nameSpecies = currentAttribute.getNodeValue();
77                              if (nameSpecies == null) {
78                                  nameSpecies = "";
79                              }
80                          } // if (currentAttributeName.equals("name")) ...
81                      } // if (namespaceURI == null || SpecificationCollection.
                            SPECIFICATION_NAMESPACE.equals(namespaceURI)) ...
82                  } // if (currentAttribute != null) ...
83              } // for (int c = 0; c < length; c++) ...
84          } // if (attributes != null) ...
85      }
86      if (nameSpecies == null) {
87          throw new SpecificationXmlLoadException("Keine referenzierte Spezies angegeben.");
88      } // if (nameSpecies == null) ...
89
90      NamedSpeciesReference result = new NamedSpeciesReference();
91      result._speciesName = nameSpecies;
92      return result;
93  } // public static NamedSpeciesReference createFromXml(org.w3c.dom.Element xmlElement) ...
94
95  /**
96   * Beendet den Xml-Ladevorgang.
97   * @throws at.techmath.boltzmann.specification.SpecificationXmlLoadException Wird geworfen,
98   *   wenn ein unbekanntes Objekt referenziert wird.
99   */
100 @Override
101 protected void finishXmlLoad() throws SpecificationXmlLoadException {
102     super.finishXmlLoad();
103     if (_speciesName != null) {
104         ISpecies parentSpecies;
105         ISpecification parentSpecification;
106         if ((parentSpecies = getParentSpecies()) == null ||
107             (parentSpecification = parentSpecies.getSpecification()) == null) {
108             throw new IllegalStateException("Parent-Referenzen der Spezies-Referenz nicht
                    richtig gesetzt.");
109         }
110         INamedSpecies species = parentSpecification.getSpecies(_speciesName);
111         if (species instanceof NamedSpecies) {
112             _species = (NamedSpecies) species;
113         } else { // if (!(species instanceof NamedSpecies)) ...
114             throw new SpecificationXmlLoadException("Spezies mit dem Namen \"" +
                        _speciesName + "\" nicht vorhanden.");
115         } // if (!(species instanceof NamedSpecies)) ...
116     } else { // if (_speciesName == null) ...
117         _species = null;
118     } // if (_nameSpecies == null) ...
119 } // protected void finishXmlLoad() throws SpecificationXmlLoadException ...
120
121 /**
122  * Gibt eine String-Repraesentation der Instanz zurueck.
123  * @return String-Repraesentation der Instanz.
124  */
125 @Override
126 public String toString() {
127     StringBuilder sb = new StringBuilder();
128     sb.append("        Spezies");
129     if (_species != null) {
130         sb.append(" \"");
131         sb.append(_species.getName());
132         sb.append("\"");
133     }
134     ILabel label = this.getLabel();
135     if (label != null) {
136         sb.append(" - Label \"");
137         sb.append(label.getName());
138         sb.append("\"");
139     } // if (label != null) ...
140     return sb.toString();
141 } // public String toString() ...
142 } // class NamedSpeciesReference extends SpeciesReference ...
```

Listing 153: Klasse *at.techmath.boltzmann.specification.Species*

```
1  package at.techmath.boltzmann.specification;
2
3  /**
4   * Abstrakte Basisklasse fuer Spezies.
5   * @author Stefan Schnabl (e0226245)
6   */
7  abstract class Species implements ISpecies {
8      /**
9       * Spezifikation, der die Spezies angehoert.
10      */
11     private Specification _parent = null;
12
13     /**
14      * Gibt die Spezifikation, der die Spezies
15      * angehoert, zurueck.
16      * @return Spezifikation der Spezies, der die Spezies angehoert.
17      */
18     public ISpecification getSpecification() {
19         return _parent;
20     } // public ISpecification getSpecification() ...
21
22     /**
23      * Setzt die Spezifikation, der die Spezies angehoert.
24      * @param parent Spezifikation, der die Spezies angehoert.
25      */
26     protected void setParent(Specification parent) {
27         _parent = parent;
28     } // protected void setParent(Specification parent) ...
29 } // abstract class Species implements ISpecies ...
```

Listing 154: Klasse *at.techmath.boltzmann.specification.SpeciesCycle*

```
1  package at.techmath.boltzmann.specification;
2
3  import org.w3c.dom.Element;
4  import org.w3c.dom.Node;
5  import org.w3c.dom.NodeList;
6
7  /**
8   * Standard-Implementierung des ISpeciesCycle-Interfaces.
9   * @author Stefan Schnabl (e0226245)
10  */
11 public class SpeciesCycle extends NamedSpecies implements ISpeciesCycle {
12     /**
13      * Name des Xml-Elements.
14      */
15     protected static final String XML_ELEMENT_NAME = "cycle";
16
17     /**
18      * Erzeugt eine neue Instanz.
19      * @param name Name der Spezies.
20      * @throws IllegalArgumentException Wird geworfen, wenn
21      *   der Name null oder leer ist.
22      */
23     public SpeciesCycle(String name) {
24         super(name);
25     } // public SpeciesCycle(String name) ...
26
27     /**
28      * Referenz auf die Spezies, von
29      * der Zyklen gebildet werden sollen.
30      */
31     private SpeciesReference _speciesParameter = null;
32
33     /**
34      * Gibt eine Referenz auf die Spezies zurueck,
35      * von der Zyklen gebildet werden sollen.
36      * @return Referenz auf die Spezies, von
37      *   der Zyklen gebildet werden sollen.
38      */
39     public ISpeciesReference getSpeciesParameter() {
40         return _speciesParameter;
41     } // public ISpeciesReference getSpeciesParameter() ...
42
43     /**
44      * Erzeugt aus dem angegebenen Xml-Element eine SpeciesCycle-Instanz.
45      * @param name Name der Spezies.
46      * @param xmlElement Xml-Element, aus dem die Instanz erzeugt werden soll.
47      * @return SpeciesCycle-Instanz, die aus dem angegebenen Xml-Element erzeugt wurde.
48      * @throws at.techmath.boltzmann.specification.SpecificationXmlLoadException Wird geworfen,
49      *   wenn die Struktur des uebergebenen Xml-Elements fehlerhaft ist.
50      * @throws IllegalArgumentException Wird geworfen, wenn das uebergebene Xml-Element
51      *   null ist oder nicht vom erwarteten Typ ist, bzw. wenn der Name ungueltig ist.
52      */
53     public static SpeciesCycle createFromXml(String name, org.w3c.dom.Element xmlElement)
54       throws SpecificationXmlLoadException {
55         if (xmlElement == null) throw new IllegalArgumentException("xmlElement = null.");
56         String elementName = xmlElement.getLocalName();
57         if (elementName == null) { elementName = ""; } else { elementName = elementName.trim().
               toLowerCase(); }
58         if (!elementName.equals(XML_ELEMENT_NAME) ||
59             !SpecificationCollection.SPECIFICATION_NAMESPACE.equals(xmlElement.getNamespaceURI
                   ())) {
60             throw new IllegalArgumentException("\"" + XML_ELEMENT_NAME + "\"-Element erwartet."
                   );
61         }
62         SpeciesCycle result = new SpeciesCycle(name);
63         // Parameter-Spezies-Referenz laden
64         {
65             NodeList childNodes = xmlElement.getChildNodes();
66             if (childNodes != null) {
67                 int length = childNodes.getLength();
68                 for (int c = 0; c < length; c++) {
69                     Node currentChild = childNodes.item(c);
70                     if (currentChild != null && currentChild.getNodeType() == Node.ELEMENT_NODE
                           && currentChild instanceof Element) {
71                         Element currentElement = (Element) currentChild;
72                         if (SpecificationCollection.SPECIFICATION_NAMESPACE.equals(
                               currentElement.getNamespaceURI())) {
73                             String currentElementName = currentElement.getLocalName();
```

```
74                              if (currentElementName == null) { currentElementName = ""; } else {
                                    currentElementName = currentElementName.trim().toLowerCase();
                                    }
75                              if (SpeciesReference.isKnownSpeciesReferenceElement(
                                    currentElementName)) {
76                                  if (result._speciesParameter != null) {
77                                      throw new SpecificationXmlLoadException("Parameter-Spezies
                                            der Cycle-Spezies mehrmals angegeben.");
78                                  } // if (result._speciesParameter != null) ...
79                                  SpeciesReference speciesReference = SpeciesReference.
                                        createFromXml(currentElement);
80                                  if (speciesReference instanceof SpeciesOneReference) {
81                                      throw new SpecificationXmlLoadException("Die One-Spezies
                                            darf nicht in die Cycle-Spezies eingesetzt werden.");
82                                  } // if (speciesReference instanceof SpeciesOneReference) ...
83                                  result._speciesParameter = speciesReference;
84                                  speciesReference.setParent(result);
85                              } // if (SpeciesReference.isKnownSpeciesReferenceElement(
                                    currentElementName)) ...
86                          } // if (SpecificationCollection.SPECIFICATION_NAMESPACE.equals(
                                currentElement.getNamespaceURI())) ...
87                      } // if (currentChild != null && currentChild.getNodeType() == Node.
                            ELEMENT_NODE && currentChild instanceof Element) ...
88                  } // for (int c = 0; c < length; c++) ...
89              } // if (childNodes != null) ...
90          }
91          if (result._speciesParameter == null) {
92              throw new SpecificationXmlLoadException("Keine Parameter-Spezies fuer die Cycle-
                    Spezies angegeben.");
93          }
94          return result;
95      } // public static SpeciesCycle createFromXml(String name, org.w3c.dom.Element xmlElement)
            ...
96
97      /**
98       * Beendet den Xml-Ladevorgang.
99       * @throws at.techmath.boltzmann.specification.SpecificationXmlLoadException Wird geworfen,
100      *   wenn ein unbekanntes Objekt referenziert wird.
101      */
102     @Override
103     protected void finishXmlLoad() throws SpecificationXmlLoadException {
104         if (_speciesParameter != null) {
105             _speciesParameter.finishXmlLoad();
106         } // if (_speciesParameter != null) ...
107     } // protected void finishXmlLoad() throws SpecificationXmlLoadException ...
108
109     /**
110      * Gibt eine String-Repraesentation der Instanz zurueck.
111      * @return String-Repraesentation der Instanz.
112      */
113     @Override
114     public String toString() {
115         StringBuilder sb = new StringBuilder();
116         sb.append("      ");
117         sb.append("Spezies \"");
118         sb.append(getName());
119         sb.append("\" = Cycle\n");
120         if (_speciesParameter != null) {
121             sb.append(_speciesParameter.toString());
122             sb.append("\n");
123         } // if (_speciesParameter != null) ...
124         return sb.toString();
125     } // public String toString() ...
126 } // public class SpeciesCycle extends NamedSpecies implements ISpeciesCycle ...
```

Listing 155: Klasse *at.techmath.boltzmann.specification.SpeciesOne*

```
1 package at.techmath.boltzmann.specification;
2
3 /**
4  * Spezies der Charakteristik der leeren Menge.
5  * @author Stefan Schnabl (e0226245)
6  */
7 class SpeciesOne extends Species implements ISpeciesOne {
8 } // class SpeciesOne extends Species implements ISpeciesOne ...
```

Listing 156: Klasse *at.techmath.boltzmann.specification.SpeciesOneReference*

```
1  package at.techmath.boltzmann.specification;
2
3  /**
4   * Referenz auf die One-Spezies.
5   * @author Stefan Schnabl (e0226245)
6   */
7  class SpeciesOneReference extends SpeciesReference {
8      /**
9       * Name des Xml-Elements.
10      */
11     protected static final String XML_ELEMENT_NAME = "one";
12
13     /**
14      * Spezies, die referenziert wird.
15      */
16     private Species _species = null;
17
18     /**
19      * Gibt die Spezies zurueck, die referenziert wird.
20      * @return Spezies, die referenziert wird.
21      */
22     public ISpecies getSpecies() {
23         return _species;
24     } // public ISpecies getSpecies() ...
25
26     /**
27      * Erzeugt aus dem uebergebenen Xml-Element eine One-Species-Referenz.
28      * @param xmlElement Xml-Element, anhand dessen die Instanz erzeugt werden soll.
29      * @return One-Species-Referenz, die aus dem uebergebenen Xml-Element erzeugt wurde.
30      * @throw InvalidArgumentException Wird geworfen, wenn das angegebene Xml-Element
31      *   null ist oder nicht vom erwarteten Typ ist.
32      */
33     public static SpeciesOneReference createFromXml(org.w3c.dom.Element xmlElement) {
34         if (xmlElement == null) throw new IllegalArgumentException("xmlElement = null");
35         String elementName = xmlElement.getLocalName();
36         if (elementName == null) { elementName = ""; } else { elementName = elementName.trim().
               toLowerCase(); }
37         if (!XML_ELEMENT_NAME.equals(elementName) ||
38             !SpecificationCollection.SPECIFICATION_NAMESPACE.equals(xmlElement.getNamespaceURI
                   ())) {
39             throw new IllegalArgumentException("\"" + XML_ELEMENT_NAME +"\"-Element erwartet.")
                   ;
40         }
41         return new SpeciesOneReference();
42     } // public static SpeciesOneReference createFromXml(org.w3c.dom.Element xmlElement) ...
43
44     /**
45      * Beendet den Xml-Ladevorgang.
46      * @throws at.techmath.boltzmann.specification.SpecificationXmlLoadException Wird geworfen,
47      *   wenn ein unbekanntes Objekt referenziert wird.
48      */
49     @Override
50     protected void finishXmlLoad() throws SpecificationXmlLoadException {
51         super.finishXmlLoad();
52         ISpecies parentSpecies;
53         ISpecification specification;
54         if ((parentSpecies = getParentSpecies()) == null ||
55             (specification = parentSpecies.getSpecification()) == null) {
56             throw new IllegalStateException("Parent-Referenzen der Spezies-Referenz nicht
                   richtig gesetzt.");
57         }
58         ISpecies referencedSpecies = specification.getSpeciesOne();
59         if (referencedSpecies instanceof Species) {
60             _species = (Species) referencedSpecies;
61         } else { // if (!referencedSpecies instanceof Species)) ...
62             throw new IllegalStateException("One-Spezies nicht gefunden.");
63         } // if (!referencedSpecies instanceof Species)) ...
64     } // protected void finishXmlLoad() throws SpecificationXmlLoadException ...
65
66     /**
67      * Gibt eine String-Repraesentation der Instanz zurueck.
68      * @return String-Repraesentation der Instanz.
69      */
70     @Override
71     public String toString() {
72         StringBuilder sb = new StringBuilder();
73         sb.append("        One");
74         ILabel label = this.getLabel();
75         if (label != null) {
```

```
76              sb.append(" - Label \"");
77              sb.append(label.getName());
78              sb.append("\"");
79          } // if (label != null) ...
80          return sb.toString();
81      } // public String toString() ...
82  } // class SpeciesOneReference extends SpeciesReference ...
```

Listing 157: Klasse *at.techmath.boltzmann.specification.SpeciesProduct*

```
1  package at.techmath.boltzmann.specification;
2
3  import java.util.Vector;
4
5  import org.w3c.dom.Element;
6  import org.w3c.dom.Node;
7  import org.w3c.dom.NodeList;
8
9  /**
10  * Standard-Implementierung des ISpeciesProduct-Interfaces.
11  * @author Stefan Schnabl (e0226245)
12  */
13 public class SpeciesProduct extends NamedSpecies implements ISpeciesProduct {
14     /**
15      * Name des Xml-Elements.
16      */
17     protected static final String XML_ELEMENT_NAME = "product";
18
19     /**
20      * Erzeugt eine neue Instanz.
21      * @param name Name der Spezies.
22      * @throws IllegalArgumentException Wird geworfen, wenn
23      *   der Name null oder leer ist.
24      */
25     public SpeciesProduct(String name) {
26         super(name);
27     } // public SpeciesProduct(String name) ...
28
29     /**
30      * Liste der Faktoren.
31      */
32     private Vector<SpeciesReference> _factorList = new Vector<SpeciesReference>();
33
34     /**
35      * Gibt die Anzahl der Faktoren zurueck.
36      * @return Anzahl der Faktoren.
37      */
38     public int getFactorCount() {
39         return _factorList.size();
40     } // public int getFactorCount() ...
41
42     /**
43      * Gibt den Faktor mit dem angegebenen Index zurueck.
44      * Der Index muss zwischen 0 und getFactorCount() - 1 liegen.
45      * @param index Index des gewuenschten Faktoren.
46      * @return Faktor mit dem angegebenen Index.
47      * @throws IndexOutOfBoundsException Der angegebene Index
48      *   liegt ausserhalb des gueltigen Bereichs.
49      */
50     public ISpeciesReference getFactorIndex(int index) {
51         if (index < 0 || index >= _factorList.size()) {
52             throw new IndexOutOfBoundsException("Der angegebene Index liegt ausserhalb des
                   gueltigen Bereichs.");
53         }
54         return _factorList.elementAt(index);
55     } // public ISpeciesReference getFactorIndex(int index) ...
56
57     /**
58      * Erzeugt aus dem angegebenen Xml-Element eine SpeciesProduct-Instanz.
59      * @param name Name der Spezies.
60      * @param xmlElement Xml-Element, aus dem die Instanz erzeugt werden soll.
61      * @return SpeciesProduct-Instanz, die aus dem angegebenen Xml-Element erzeugt wurde.
62      * @throws at.techmath.boltzmann.specification.SpecificationXmlLoadException Wird geworfen,
63      *   wenn die Struktur des uebergebenen Xml-Elements fehlerhaft ist.
64      * @throws IllegalArgumentException Wird geworfen, wenn das uebergebene Xml-Element
65      *   null ist oder nicht vom erwarteten Typ ist, bzw. wenn der Name ungueltig ist.
66      */
67     public static SpeciesProduct createFromXml(String name, org.w3c.dom.Element xmlElement)
68       throws SpecificationXmlLoadException {
69         if (xmlElement == null) throw new IllegalArgumentException("xmlElement = null.");
70         String elementName = xmlElement.getLocalName();
71         if (elementName == null) { elementName = ""; } else { elementName = elementName.trim().
               toLowerCase(); }
72         if (!elementName.equals(XML_ELEMENT_NAME) ||
73             !SpecificationCollection.SPECIFICATION_NAMESPACE.equals(xmlElement.getNamespaceURI
                   ())) {
74             throw new IllegalArgumentException("\"" + XML_ELEMENT_NAME + "\"-Element erwartet."
                   );
75         }
```

```
76          SpeciesProduct result = new SpeciesProduct(name);
77          // Parameter-Spezies-Referenzen laden
78          {
79              NodeList childNodes = xmlElement.getChildNodes();
80              if (childNodes != null) {
81                  int length = childNodes.getLength();
82                  for (int c = 0; c < length; c++) {
83                      Node currentChild = childNodes.item(c);
84                      if (currentChild != null && currentChild.getNodeType() == Node.ELEMENT_NODE
                            && currentChild instanceof Element) {
85                          Element currentElement = (Element) currentChild;
86                          if (SpecificationCollection.SPECIFICATION_NAMESPACE.equals(
                                currentElement.getNamespaceURI())) {
87                              String currentElementName = currentElement.getLocalName();
88                              if (currentElementName == null) { currentElementName = ""; } else {
                                    currentElementName = currentElementName.trim().toLowerCase();
                                    }
89                              if (SpeciesReference.isKnownSpeciesReferenceElement(
                                    currentElementName)) {
90                                  SpeciesReference currentReference = SpeciesReference.
                                        createFromXml(currentElement);
91                                  result._factorList.add(currentReference);
92                                  currentReference.setParent(result);
93                              } // if (SpeciesReference.isKnownSpeciesReferenceElement(
                                    currentElementName)) ...
94                          } // if (SpecificationCollection.SPECIFICATION_NAMESPACE.equals(
                                currentElement.getNamespaceURI())) ...
95                      } // if (currentChild != null && currentChild.getNodeType() == Node.
                            ELEMENT_NODE && currentChild instanceof Element) ...
96                  } // for (int c = 0; c < length; c++) ...
97              } // if (childNodes != null) ...
98          }
99          if (result._factorList.size() <= 0) {
100             throw new SpecificationXmlLoadException("Fuer Produkt-Spezies muss mindestens eine
                    Produkt-Spezies angegeben werden.");
101         }
102         return result;
103     } // public static SpeciesProduct createFromXml(String name, org.w3c.dom.Element xmlElement
        ) ...
104
105     /**
106      * Beendet den Xml-Ladevorgang.
107      * @throws at.techmath.boltzmann.specification.SpecificationXmlLoadException Wird geworfen,
108      *   wenn ein unbekanntes Objekt referenziert wird.
109      */
110     @Override
111     protected void finishXmlLoad() throws SpecificationXmlLoadException {
112         for (SpeciesReference speciesReference : _factorList) {
113             speciesReference.finishXmlLoad();
114         } // for (SpeciesReference speciesReference : _factorList) ...
115     } // protected void finishXmlLoad() throws SpecificationXmlLoadException ...
116
117     /**
118      * Gibt eine String-Repraesentation der Instanz zurueck.
119      * @return String-Repraesentation der Instanz.
120      */
121     @Override
122     public String toString() {
123         StringBuilder sb = new StringBuilder();
124         sb.append("      ");
125         sb.append("Spezies \"");
126         sb.append(getName());
127         sb.append("\" = Produkt\n");
128         for (SpeciesReference speciesReference : _factorList) {
129             sb.append(speciesReference.toString()).append("\n");
130         } // for (SpeciesReference speciesReference : _factorList) ...
131         return sb.toString();
132     } // public String toString() ...
133 } // public class SpeciesProduct extends NamedSpecies implements ISpeciesProduct ...
```

Listing 158: Klasse *at.techmath.boltzmann.specification.SpeciesReference*

```
1  package at.techmath.boltzmann.specification;
2
3  import org.w3c.dom.Node;
4  import org.w3c.dom.NamedNodeMap;
5
6  /**
7   * Basisklasse fuer die Spezies-Referenzen.
8   * @author Stefan Schnabl (e0226245)
9   */
10 abstract class SpeciesReference implements ISpeciesReference {
11     /**
12      * Spezies, von der die Referenz verwendet wird.
13      */
14     private Species _parentSpecies = null;
15
16     /**
17      * Gibt die Spezies zurueck, von der die
18      * Referenz verwendet wird.
19      * @return Spezies, von der die Referenz verwendet wird.
20      */
21     public ISpecies getParentSpecies() {
22         return _parentSpecies;
23     } // public ISpecies getParentSpecies() ...
24
25     /**
26      * Setzt die Spezies, von der die Referenz verwendet wird.
27      * @param parent Spezies, von der die Referenz verwendet wird.
28      */
29     protected void setParent(Species parent) {
30         _parentSpecies = parent;
31     } // protected void setParent(Species parent) ...
32
33     /**
34      * Label, das der Referenz zugeordnet ist.
35      */
36     private Label _label = null;
37
38     /**
39      * Name des Labels, das der Referenz zugeordnet ist.
40      */
41     private String _labelName = null;
42
43     /**
44      * Setzt den Namen des Labels, das der Referenz zugeordnet ist.
45      * @param labelName Name des Labels, das der Referenz zugeordnet ist.
46      */
47     protected void setLabelName(String labelName) {
48         _labelName = labelName;
49     } // protected void setLabelName(String labelName) ...
50
51     /**
52      * Gibt das Label, das der Referenz zugeordnet ist,
53      * zurueck.
54      * @return Label, das der Referenz zugeordnet ist,
55      *   falls vorhanden, ansonsten null.
56      */
57     public ILabel getLabel() {
58         return _label;
59     } // public ILabel getLabel() ...
60
61     /**
62      * Gibt die Spezies zurueck, die referenziert wird.
63      * @return Spezies, die referenziert wird.
64      */
65     public abstract ISpecies getSpecies();
66
67     /**
68      * Erzeugt aus dem angegebenen Xml-Element die entsprechende
69      * Spezies-Referenz.
70      * @param xmlElement Xml-Element, aus dem die Spezies-Referenz erzeugt werden soll.
71      * @return Spezies-Referenz, die aus dem angegebenen Xml-Element erzeugt wurde.
72      * @throws InvalidArgumentException Wird geworfen, wenn das uebergebene
73      *   Xml-Element null ist.
74      * @throws at.techmath.boltzmann.specification.SpecificationXmlLoadException Wird geworfen,
            wenn
75      *   das Xml-Element nicht die erwartete Struktur besitzt.
76      */
77     public static SpeciesReference createFromXml(org.w3c.dom.Element xmlElement)
78       throws SpecificationXmlLoadException {
```

```
79          if (xmlElement == null) throw new IllegalArgumentException("xmlElement = null");
80          // Namen des referenzierten Labels laden
81          String nameLabel = null;
82          {
83              NamedNodeMap attributes = xmlElement.getAttributes();
84              if (attributes != null) {
85                  int length = attributes.getLength();
86                  for (int c = 0; c < length; c++) {
87                      Node currentAttribute = attributes.item(c);
88                      if (currentAttribute != null) {
89                          String namespaceURI = currentAttribute.getNamespaceURI();
90                          if (namespaceURI == null || SpecificationCollection.
                                SPECIFICATION_NAMESPACE.equals(namespaceURI)) {
91                              String currentAttributeName = currentAttribute.getLocalName();
92                              if (currentAttributeName == null) {
93                                  currentAttributeName = "";
94                              } else { // if (currentAttributeName != null) ...
95                                  currentAttributeName = currentAttributeName.trim().toLowerCase
                                        ();
96                              } // if (currentAttributeName != null) ...
97                              if (currentAttributeName.equals("label")) {
98                                  if (nameLabel != null) {
99                                      throw new SpecificationXmlLoadException("Name des
                                            referenzierten Labels mehrmals angegeben.");
100                                 } // if (nameLabel != null) ...
101                                 nameLabel = currentAttribute.getNodeValue();
102                                 if (nameLabel == null) {
103                                     nameLabel = "";
104                                 } // if (nameLabel == null) ...
105                             } // if (currentAttributeName.equals("label")) ...
106                         } // if (namespaceURI == null || SpecificationCollection.
                                SPECIFICATION_NAMESPACE.equals(namespaceURI)) ...
107                     } // if (currentAttribute != null) ...
108                 } // for (int c = 0; c < length; c++0 ...
109             } // if (attributes != null) ...
110         }
111         SpeciesReference result = null;
112         if (SpecificationCollection.SPECIFICATION_NAMESPACE.equals(xmlElement.getNamespaceURI()
                )) {
113             String elementName = xmlElement.getLocalName();
114             if (elementName == null) {
115                 elementName = "";
116             } else { // if (elementName != null) ...
117                 elementName = elementName.trim().toLowerCase();
118             } // if (elementName != null) ...
119             if (elementName.equals(NamedSpeciesReference.XML_ELEMENT_NAME)) {
120                 result = NamedSpeciesReference.createFromXml(xmlElement);
121             } else if (elementName.equals(SpeciesSingletonReference.XML_ELEMENT_NAME)) {
122                 result = SpeciesSingletonReference.createFromXml(xmlElement);
123             } else if (elementName.equals(SpeciesOneReference.XML_ELEMENT_NAME)) {
124                 result = SpeciesOneReference.createFromXml(xmlElement);
125             }
126         } // if (SpecificationCollection.SPECIFICATION_NAMESPACE.equals(xmlElement.
                getNamespaceURI())) ...
127         if (result == null) {
128             throw new SpecificationXmlLoadException("Unbekannter Spezies-Referenz-Typ.");
129         } // if (result == null) ...
130         result.setLabelName(nameLabel);
131         return result;
132     } // public static SpeciesReference createFromXml(org.w3c.dom.Element xmlElement) ...
133
134     /**
135      * Ermittelt, ob der uebergebene Xml-Element-Name
136      * eine bekannte Spezies-Referenz darstellt.
137      * @param name Xml-Element-Name, der ueberprueft werden soll.
138      * @return true, falls es sich um einen bekannten Xml-Element-Namen
139      *   handelt, ansonsten false.
140      */
141     protected static boolean isKnownSpeciesReferenceElement(String name) {
142         String normName = name == null ? "" : name.trim().toLowerCase();
143         return
144             NamedSpeciesReference.XML_ELEMENT_NAME.equals(normName) ||
145             SpeciesSingletonReference.XML_ELEMENT_NAME.equals(normName) ||
146             SpeciesOneReference.XML_ELEMENT_NAME.equals(normName);
147     } // protected static isKnownSpeciesReferenceElement(String name) ...
148
149     /**
150      * Beendet den Xml-Ladevorgang.
151      * @throws at.techmath.boltzmann.specification.SpecificationXmlLoadException Wird geworfen,
152      *   wenn ein Label referenziert wird, das nicht vorhanden ist.
```

```
153      */
154     protected void finishXmlLoad() throws SpecificationXmlLoadException {
155         // Label ermitteln
156         if (_labelName != null) {
157             ISpecies parentSpecies;
158             ISpecification parentSpecification;
159             ILabelCollection labelCollection;
160             if ((parentSpecies = getParentSpecies()) == null ||
161                 (parentSpecification = parentSpecies.getSpecification()) == null ||
162                 (labelCollection = parentSpecification.getLabelCollection()) == null) {
163                 throw new IllegalStateException("Parent-Referenzen der Spezies-Referenz nicht
                        richtig gesetzt.");
164             }
165             ILabel label = labelCollection.getLabel(_labelName);
166             if (label instanceof Label) {
167                 _label = (Label) label;
168             } else { // if (!(label instanceof Label) ...
169                 throw new SpecificationXmlLoadException("Label mit dem Namen \"" + _labelName +
                        "\" nicht vorhanden.");
170             } // if (!(label instanceof Label) ...
171         } else { // if (_labelName == null) ...
172             _label = null;
173         } // if (_labelName == null) ...
174     } // protected void finishXmlLoad() throws SpecificationXmlLoadException ...
175 } // abstract class SpeciesReference implements ISpeciesReference ...
```

Listing 159: Klasse *at.techmath.boltzmann.specification.SpeciesSequence*

```
1  package at.techmath.boltzmann.specification;
2
3  import org.w3c.dom.Element;
4  import org.w3c.dom.Node;
5  import org.w3c.dom.NodeList;
6
7  /**
8   * Standard-Implementierung des ISpeciesSequence-Interfaces.
9   * @author Stefan Schnabl (e0226245)
10  */
11 public class SpeciesSequence extends NamedSpecies implements ISpeciesSequence {
12     /**
13      * Name des Xml-Elements.
14      */
15     protected static final String XML_ELEMENT_NAME = "sequence";
16
17     /**
18      * Erzeugt eine neue Instanz.
19      * @param name Name der Spezies.
20      * @throws IllegalArgumentException Wird geworfen, wenn
21      *   der Name null oder leer ist.
22      */
23     public SpeciesSequence(String name) {
24         super(name);
25     } // public SpeciesSequence(String name) ...
26
27     /**
28      * Referenz auf die Spezies, von
29      * der endliche Folgen gebildet werden sollen.
30      */
31     private SpeciesReference _speciesParameter = null;
32
33     /**
34      * Gibt eine Referenz auf die Spezies zurueck,
35      * von der endliche Folgen gebildet werden sollen.
36      * @return Referenz auf die Spezies, von der
37      *   endliche Folgen gebildet werden sollen.
38      */
39     public ISpeciesReference getSpeciesParameter() {
40         return _speciesParameter;
41     } // public ISpeciesReference getSpeciesParameter() ...
42
43     /**
44      * Erzeugt aus dem angegebenen Xml-Element eine SpeciesSequence-Instanz.
45      * @param name Name der Spezies.
46      * @param xmlElement Xml-Element, aus dem die Instanz erzeugt werden soll.
47      * @return SpeciesSequence-Instanz, die aus dem angegebenen Xml-Element erzeugt wurde.
48      * @throws at.techmath.boltzmann.specification.SpecificationXmlLoadException Wird geworfen,
49      *   wenn die Struktur des uebergebenen Xml-Elements fehlerhaft ist.
50      * @throws IllegalArgumentException Wird geworfen, wenn das uebergebene Xml-Element
51      *   null ist oder nicht vom erwarteten Typ ist, bzw. wenn der Name ungueltig ist.
52      */
53     public static SpeciesSequence createFromXml(String name, org.w3c.dom.Element xmlElement)
54       throws SpecificationXmlLoadException {
55         if (xmlElement == null) throw new IllegalArgumentException("xmlElement = null.");
56         String elementName = xmlElement.getLocalName();
57         if (elementName == null) { elementName = ""; } else { elementName = elementName.trim().
               toLowerCase(); }
58         if (!elementName.equals(XML_ELEMENT_NAME) ||
59             !SpecificationCollection.SPECIFICATION_NAMESPACE.equals(xmlElement.getNamespaceURI
                   ())) {
60             throw new IllegalArgumentException("\"" + XML_ELEMENT_NAME + "\"-Element erwartet."
                   );
61         }
62         SpeciesSequence result = new SpeciesSequence(name);
63         // Parameter-Spezies-Referenz laden
64         {
65             NodeList childNodes = xmlElement.getChildNodes();
66             if (childNodes != null) {
67                 int length = childNodes.getLength();
68                 for (int c = 0; c < length; c++) {
69                     Node currentChild = childNodes.item(c);
70                     if (currentChild != null && currentChild.getNodeType() == Node.ELEMENT_NODE
                           && currentChild instanceof Element) {
71                         Element currentElement = (Element) currentChild;
72                         if (SpecificationCollection.SPECIFICATION_NAMESPACE.equals(
                               currentElement.getNamespaceURI())) {
73                             String currentElementName = currentElement.getLocalName();
```

```
74                              if (currentElementName == null) { currentElementName = ""; } else {
                                    currentElementName = currentElementName.trim().toLowerCase();
                                    }
75                              if (SpeciesReference.isKnownSpeciesReferenceElement(
                                    currentElementName)) {
76                                  if (result._speciesParameter != null) {
77                                      throw new SpecificationXmlLoadException("Parameter-Spezies
                                            der Sequence-Spezies mehrmals angegeben.");
78                                  } /// if (result._speciesParameter != null) ...
79                                  SpeciesReference speciesReference = SpeciesReference.
                                        createFromXml(currentElement);
80                                  if (speciesReference instanceof SpeciesOneReference) {
81                                      throw new SpecificationXmlLoadException("Die One-Spezies
                                            darf nicht in die Sequence-Spezies eingesetzt werden."
                                            );
82                                  } // if (speciesReference instanceof SpeciesOneReference) ...
83                                  result._speciesParameter = speciesReference;
84                                  speciesReference.setParent(result);
85                              } // if (SpeciesReference.isKnownSpeciesReferenceElement(
                                    currentElementName)) ...
86                          } // if (SpecificationCollection.SPECIFICATION_NAMESPACE.equals(
                                currentElement.getNamespaceURI())) ...
87                      } // if (currentChild != null && currentChild.getNodeType() == Node.
                            ELEMENT_NODE && currentChild instanceof Element) ...
88                  } // for (int c = 0; c < length; c++) ...
89              } // if (childNodes != null) ...
90          }
91          if (result._speciesParameter == null) {
92              throw new SpecificationXmlLoadException("Keine Parameter-Spezies fuer die Sequence-
                    Spezies angegeben.");
93          }
94          return result;
95      } // public static SpeciesSequence createFromXml(String name, org.w3c.dom.Element
            xmlElement) ...
96
97      /**
98       * Beendet den Xml-Ladevorgang.
99       * @throws at.techmath.boltzmann.specification.SpecificationXmlLoadException Wird geworfen,
100      *   wenn ein unbekanntes Objekt referenziert wird.
101      */
102     @Override
103     protected void finishXmlLoad() throws SpecificationXmlLoadException {
104         if (_speciesParameter != null) {
105             _speciesParameter.finishXmlLoad();
106         } // if (_speciesParameter != null) ...
107     } // protected void finishXmlLoad() throws SpecificationXmlLoadException ...
108
109     /**
110      * Gibt eine String-Repraesentation der Instanz zurueck.
111      * @return String-Repraesentation der Instanz.
112      */
113     @Override
114     public String toString() {
115         StringBuilder sb = new StringBuilder();
116         sb.append("      ");
117         sb.append("Spezies \"");
118         sb.append(getName());
119         sb.append("\" = Sequence\n");
120         if (_speciesParameter != null) {
121             sb.append(_speciesParameter.toString());
122             sb.append("\n");
123         } // if (_speciesParameter != null) ...
124         return sb.toString();
125     } // public String toString() ...
126 } // public class SpeciesSequence extends NamedSpecies implements ISpeciesSequence ...
```

Listing 160: Klasse *at.techmath.boltzmann.specification.SpeciesSet*

```
1  package at.techmath.boltzmann.specification;
2
3  import org.w3c.dom.Node;
4  import org.w3c.dom.NodeList;
5  import org.w3c.dom.Element;
6
7  /**
8   * Standard-Implementierung des ISpeciesSet-Interfaces.
9   * @author Stefan Schnabl (e0226245)
10  */
11 class SpeciesSet extends NamedSpecies implements ISpeciesSet {
12     /**
13      * Name des Xml-Elements.
14      */
15     protected static final String XML_ELEMENT_NAME = "set";
16
17     /**
18      * Erzeugt eine neue Instanz.
19      * @param name Name der Spezies.
20      * @throws IllegalArgumentException Wird geworfen, wenn
21      *   der Name null oder leer ist.
22      */
23     public SpeciesSet(String name) {
24         super(name);
25     } // public SpeciesSet(String name) ...
26
27     /**
28      * Referenz auf die Spezies, von
29      * der Mengen gebildet werden sollen.
30      */
31     private SpeciesReference _speciesParameter = null;
32
33     /**
34      * Gibt eine Referenz auf die Spezies zurueck,
35      * von der Mengen gebildet werden sollen.
36      * @return Referenz auf die Spezies, von der
37      *   Mengen gebildet werden sollen.
38      */
39     public ISpeciesReference getSpeciesParameter() {
40         return _speciesParameter;
41     } // public ISpeciesReference getSpeciesParameter() ...
42
43     /**
44      * Erzeugt aus dem angegebenen Xml-Element eine SpeciesSet-Instanz.
45      * @param name Name der Spezies.
46      * @param xmlElement Xml-Element, aus dem die Instanz erzeugt werden soll.
47      * @return SpeciesSet-Instanz, die aus dem angegebenen Xml-Element erzeugt wurde.
48      * @throws at.techmath.boltzmann.specification.SpecificationXmlLoadException Wird geworfen,
49      *   wenn die Struktur des uebergebenen Xml-Elements fehlerhaft ist.
50      * @throws IllegalArgumentException Wird geworfen, wenn das uebergebene Xml-Element
51      *   null ist oder nicht vom erwarteten Typ ist, bzw. wenn der Name ungueltig ist.
52      */
53     public static SpeciesSet createFromXml(String name, org.w3c.dom.Element xmlElement)
54       throws SpecificationXmlLoadException {
55         if (xmlElement == null) throw new IllegalArgumentException("xmlElement = null.");
56         String elementName = xmlElement.getLocalName();
57         if (elementName == null) { elementName = ""; } else { elementName = elementName.trim().
               toLowerCase(); }
58         if (!elementName.equals(XML_ELEMENT_NAME) ||
59             !SpecificationCollection.SPECIFICATION_NAMESPACE.equals(xmlElement.getNamespaceURI
                   ())) {
60             throw new IllegalArgumentException("\"" + XML_ELEMENT_NAME + "\"-Element erwartet."
                   );
61         }
62         SpeciesSet result = new SpeciesSet(name);
63         // Parameter-Spezies-Referenz laden
64         {
65             NodeList childNodes = xmlElement.getChildNodes();
66             if (childNodes != null) {
67                 int length = childNodes.getLength();
68                 for (int c = 0; c < length; c++) {
69                     Node currentChild = childNodes.item(c);
70                     if (currentChild != null && currentChild.getNodeType() == Node.ELEMENT_NODE
                           && currentChild instanceof Element) {
71                         Element currentElement = (Element) currentChild;
72                         if (SpecificationCollection.SPECIFICATION_NAMESPACE.equals(
                               currentElement.getNamespaceURI())) {
73                             String currentElementName = currentElement.getLocalName();
```

```
74                              if (currentElementName == null) { currentElementName = ""; } else {
                                    currentElementName = currentElementName.trim().toLowerCase();
                                    }
75                              if (SpeciesReference.isKnownSpeciesReferenceElement(
                                    currentElementName)) {
76                                  if (result._speciesParameter != null) {
77                                      throw new SpecificationXmlLoadException("Parameter-Spezies
                                            der Set-Spezies mehrmals angegeben.");
78                                  } // if (result._speciesParameter != null) ...
79                                  SpeciesReference speciesReference = SpeciesReference.
                                        createFromXml(currentElement);
80                                  if (speciesReference instanceof SpeciesOneReference) {
81                                      throw new SpecificationXmlLoadException("Die One-Spezies
                                            darf nicht in die Set-Spezies eingesetzt werden.");
82                                  } // if (speciesReference instanceof SpeciesOneReference) ...
83                                  result._speciesParameter = speciesReference;
84                                  speciesReference.setParent(result);
85                              } // if (SpeciesReference.isKnownSpeciesReferenceElement(
                                    currentElementName)) ...
86                          } // if (SpecificationCollection.SPECIFICATION_NAMESPACE.equals(
                                currentElement.getNamespaceURI())) ...
87                      } // if (currentChild != null && currentChild.getNodeType() == Node.
                            ELEMENT_NODE && currentChild instanceof Element) ...
88                  } // for (int c = 0; c < length; c++) ...
89              } // if (childNodes != null) ...
90          }
91          if (result._speciesParameter == null) {
92              throw new SpecificationXmlLoadException("Keine Parameter-Spezies fuer die Set-
                    Spezies angegeben.");
93          }
94          return result;
95      } // public static SpeciesSet createFromXml(String name, org.w3c.dom.Element xmlElement)
            ...
96
97      /**
98       * Beendet den Xml-Ladevorgang.
99       * @throws at.techmath.boltzmann.specification.SpecificationXmlLoadException Wird geworfen,
100      *   wenn ein unbekanntes Objekt referenziert wird.
101      */
102     @Override
103     protected void finishXmlLoad() throws SpecificationXmlLoadException {
104         if (_speciesParameter != null) {
105             _speciesParameter.finishXmlLoad();
106         } // if (_speciesParameter != null) ...
107     } // protected void finishXmlLoad() throws SpecificationXmlLoadException ...
108
109     /**
110      * Gibt eine String-Repraesentation der Instanz zurueck.
111      * @return String-Repraesentation der Instanz.
112      */
113     @Override
114     public String toString() {
115         StringBuilder sb = new StringBuilder();
116         sb.append("      ");
117         sb.append("Spezies \"");
118         sb.append(getName());
119         sb.append("\" = Set\n");
120         if (_speciesParameter != null) {
121             sb.append(_speciesParameter.toString());
122             sb.append("\n");
123         } // if (_speciesParameter != null) ...
124         return sb.toString();
125     } // public String toString() ...
126 } // class SpeciesSet extends NamedSpecies implements ISpeciesSet ...
```

Listing 161: Klasse *at.techmath.boltzmann.specification.SpeciesSingleton*

```
1 package at.techmath.boltzmann.specification;
2
3 /**
4  * Singleton-Spezies.
5  * @author Stefan Schnabl (e0226245).
6  */
7 class SpeciesSingleton extends Species implements ISpeciesSingleton {
8 } // class SpeciesSingleton extends Species implements ISpeciesSingleton ...
```

Listing 162: Klasse *at.techmath.boltzmann.specification.SpeciesSingletonReference*

```
1  package at.techmath.boltzmann.specification;
2
3  /**
4   * Referenz auf die Singleton-Spezies.
5   * @author Stefan Schnabl (e0226245)
6   */
7  class SpeciesSingletonReference extends SpeciesReference {
8      /**
9       * Name des Xml-Elements.
10      */
11     protected static final String XML_ELEMENT_NAME = "atom";
12
13     /**
14      * Spezies, die referenziert wird.
15      */
16     private Species _species = null;
17
18     /**
19      * Gibt die Spezies zurueck, die referenziert wird.
20      * @return Spezies, die referenziert wird.
21      */
22     public ISpecies getSpecies() {
23         return _species;
24     } // public ISpecies getSpecies() ...
25
26     /**
27      * Erzeugt aus dem uebergebenen Xml-Element eine Singleton-Species-Referenz.
28      * @param xmlElement Xml-Element, anhand dessen die Instanz erzeugt werden soll.
29      * @return Singleton-Species-Referenz, die aus dem uebergebenen Xml-Element erzeugt wurde.
30      * @throw InvalidArgumentException Wird geworfen, wenn das angegebene Xml-Element
31      *   null ist oder nicht vom erwarteten Typ ist.
32      */
33     public static SpeciesSingletonReference createFromXml(org.w3c.dom.Element xmlElement) {
34         if (xmlElement == null) throw new IllegalArgumentException("xmlElement = null");
35         String elementName = xmlElement.getLocalName();
36         if (elementName == null) { elementName = ""; } else { elementName = elementName.trim().
               toLowerCase(); }
37         if (!XML_ELEMENT_NAME.equals(elementName) ||
38             !SpecificationCollection.SPECIFICATION_NAMESPACE.equals(xmlElement.getNamespaceURI
                   ())) {
39             throw new IllegalArgumentException("\"" + XML_ELEMENT_NAME +"\"-Element erwartet.")
                   ;
40         }
41         return new SpeciesSingletonReference();
42     } // public static SingletonSpeciesReference createFromXml(org.w3c.dom.Element.xmlElement)
           ...
43
44     /**
45      * Beendet den Xml-Ladevorgang.
46      * @throws at.techmath.boltzmann.specification.SpecificationXmlLoadException Wird geworfen,
47      *   wenn ein unbekanntes Objekt referenziert wird.
48      */
49     @Override
50     protected void finishXmlLoad() throws SpecificationXmlLoadException {
51         super.finishXmlLoad();
52         ISpecies parentSpecies;
53         ISpecification specification;
54         if ((parentSpecies = getParentSpecies()) == null ||
55             (specification = parentSpecies.getSpecification()) == null) {
56             throw new IllegalStateException("Parent-Referenzen der Spezies-Referenz nicht
                   richtig gesetzt.");
57         }
58         ISpecies referencedSpecies = specification.getSpeciesSingleton();
59         if (referencedSpecies instanceof Species) {
60             _species = (Species) referencedSpecies;
61         } else { // if (!referencedSpecies instanceof Species)) ...
62             throw new IllegalStateException("Singleton-Spezies nicht gefunden.");
63         } // if (!referencedSpecies instanceof Species)) ...
64     } // protected void finishXmlLoad() throws SpecificationXmlLoadException ...
65
66     /**
67      * Gibt eine String-Repraesentation der Instanz zurueck.
68      * @return String-Repraesentation der Instanz.
69      */
70     @Override
71     public String toString() {
72         StringBuilder sb = new StringBuilder();
73         sb.append("        Atom");
74         ILabel label = this.getLabel();
```

```
75          if (label != null) {
76              sb.append(" - Label \"");
77              sb.append(label.getName());
78              sb.append("\"");
79          } // if (label != null) ...
80          return sb.toString();
81      } // public String toString() ...
82  } // class SingletonSpeciesReference extends SpeciesReference ...
```

Listing 163: Klasse *at.techmath.boltzmann.specification.SpeciesSum*

```
1  package at.techmath.boltzmann.specification;
2
3  import java.util.Vector;
4
5  import org.w3c.dom.Element;
6  import org.w3c.dom.Node;
7  import org.w3c.dom.NodeList;
8
9  /**
10  * Standard-Implementierung des ISpeciesSum-Interfaces.
11  * @author Stefan Schnabl (e0226245)
12  */
13 public class SpeciesSum extends NamedSpecies implements ISpeciesSum {
14     /**
15      * Name des Xml-Elements.
16      */
17     protected static final String XML_ELEMENT_NAME = "sum";
18
19     /**
20      * Erzeugt eine neue Instanz.
21      * @param name Name der Spezies.
22      * @throws IllegalArgumentException Wird geworfen, wenn
23      *   der Name null oder leer ist.
24      */
25     public SpeciesSum(String name) {
26         super(name);
27     } // public SpeciesSum(String name) ...
28
29     /**
30      * Liste der Summanden.
31      */
32     private Vector<SpeciesReference> _addendList = new Vector<SpeciesReference>();
33
34     /**
35      * Gibt die Anzahl der Summanden zurueck.
36      * @return Anzahl der Summanden.
37      */
38     public int getAddendCount() {
39         return _addendList.size();
40     } // public int getAddendCount() ...
41
42     /**
43      * Gibt den Summanden mit dem angegebenen Index zurueck.
44      * Der Index muss zwischen 0 und getAddendCount() - 1 liegen.
45      * @param index Index des gewuenschten Summanden.
46      * @return Summand mit dem angegebenen Index.
47      * @throws IndexOutOfBoundsException Der angegebene Index
48      *   liegt ausserhalb des gueltigen Bereichs.
49      */
50     public ISpeciesReference getAddendIndex(int index) {
51         if (index < 0 || index >= _addendList.size()) {
52             throw new IndexOutOfBoundsException("Der angegebene Index liegt ausserhalb des
                   gueltigen Bereichs.");
53         }
54         return _addendList.elementAt(index);
55     } // public ISpeciesReference getAddendIndex(int index) ...
56
57     /**
58      * Erzeugt aus dem angegebenen Xml-Element eine SpeciesSum-Instanz.
59      * @param name Name der Spezies.
60      * @param xmlElement Xml-Element, aus dem die Instanz erzeugt werden soll.
61      * @return SpeciesSum-Instanz, die aus dem angegebenen Xml-Element erzeugt wurde.
62      * @throws at.techmath.boltzmann.specification.SpecificationXmlLoadException Wird geworfen,
63      *   wenn die Struktur des uebergebenen Xml-Elements fehlerhaft ist.
64      * @throws IllegalArgumentException Wird geworfen, wenn das uebergebene Xml-Element
65      *   null ist oder nicht vom erwarteten Typ ist, bzw. wenn der Name ungueltig ist.
66      */
67     public static SpeciesSum createFromXml(String name, org.w3c.dom.Element xmlElement)
68       throws SpecificationXmlLoadException {
69         if (xmlElement == null) throw new IllegalArgumentException("xmlElement = null.");
70         String elementName = xmlElement.getLocalName();
71         if (elementName == null) { elementName = ""; } else { elementName = elementName.trim().
               toLowerCase(); }
72         if (!elementName.equals(XML_ELEMENT_NAME) ||
73             !SpecificationCollection.SPECIFICATION_NAMESPACE.equals(xmlElement.getNamespaceURI
                   ())) {
74             throw new IllegalArgumentException("\"" + XML_ELEMENT_NAME + "\"-Element erwartet."
                   );
75         }
```

```
76          SpeciesSum result = new SpeciesSum(name);
77          // Parameter-Spezies-Referenzen laden
78          {
79              NodeList childNodes = xmlElement.getChildNodes();
80              if (childNodes != null) {
81                  int length = childNodes.getLength();
82                  for (int c = 0; c < length; c++) {
83                      Node currentChild = childNodes.item(c);
84                      if (currentChild != null && currentChild.getNodeType() == Node.ELEMENT_NODE
                            && currentChild instanceof Element) {
85                          Element currentElement = (Element) currentChild;
86                          if (SpecificationCollection.SPECIFICATION_NAMESPACE.equals(
                                currentElement.getNamespaceURI())) {
87                              String currentElementName = currentElement.getLocalName();
88                              if (currentElementName == null) { currentElementName = ""; } else {
                                    currentElementName = currentElementName.trim().toLowerCase();
                                    }
89                              if (SpeciesReference.isKnownSpeciesReferenceElement(
                                    currentElementName)) {
90                                  SpeciesReference currentReference = SpeciesReference.
                                        createFromXml(currentElement);
91                                  result._addendList.add(currentReference);
92                                  currentReference.setParent(result);
93                              } // if (SpeciesReference.isKnownSpeciesReferenceElement(
                                    currentElementName)) ...
94                          } // if (SpecificationCollection.SPECIFICATION_NAMESPACE.equals(
                                currentElement.getNamespaceURI())) ...
95                      } // if (currentChild != null && currentChild.getNodeType() == Node.
                            ELEMENT_NODE && currentChild instanceof Element) ...
96                  } // for (int c = 0; c < length; c++) ...
97              } // if (childNodes != null) ...
98          }
99          if (result._addendList.size() <= 0) {
100             throw new SpecificationXmlLoadException("Fuer Summen-Spezies muss mindestens eine
                    Summanden-Spezies angegeben werden.");
101         }
102         return result;
103     } // public static SpeciesSum createFromXml(String name, org.w3c.dom.Element xmlElement)
          ...
104
105     /**
106      * Beendet den Xml-Ladevorgang.
107      * @throws at.techmath.boltzmann.specification.SpecificationXmlLoadException Wird geworfen,
108      *   wenn ein unbekanntes Objekt referenziert wird.
109      */
110     @Override
111     protected void finishXmlLoad() throws SpecificationXmlLoadException {
112         for (SpeciesReference speciesReference : _addendList) {
113             speciesReference.finishXmlLoad();
114         } // for (SpeciesReference speciesReference : _addendList) ...
115     } // protected void finishXmlLoad() throws SpecificationXmlLoadException ...
116
117     /**
118      * Gibt eine String-Repraesentation der Instanz zurueck.
119      * @return String-Repraesentation der Instanz.
120      */
121     @Override
122     public String toString() {
123         StringBuilder sb = new StringBuilder();
124         sb.append("      ");
125         sb.append("Spezies \"");
126         sb.append(getName());
127         sb.append("\" = Summe\n");
128         for (SpeciesReference speciesReference : _addendList) {
129             sb.append(speciesReference.toString()).append("\n");
130         } // for (SpeciesReference speciesReference : _addendList) ...
131         return sb.toString();
132     } // public String toString() ...
133 } // public class SpeciesSum extends NamedSpecies implements ISpeciesSum ...
```

Listing 164: Klasse *at.techmath.boltzmann.specification.Specification*

```
1  package at.techmath.boltzmann.specification;
2
3  import java.util.Vector;
4
5  import org.w3c.dom.*;
6
7  /**
8   * Standard-Implementierung des ISpecification-Interfaces.
9   * @author Stefan Schnabl (e0266245)
10  */
11 class Specification implements ISpecification {
12     /**
13      * Erzeugt eine neue Spezifikation.
14      * @param name Name der Spezifikation
15      * @throws IllegalArgumentException Wird geworfen, wenn
16      *   der angegebene Name null ist oder normiert eine Laenge <= 0 hat.
17      */
18     public Specification(String name) {
19         super();
20         if (name == null) throw new IllegalArgumentException("name = null.");
21         name = name.trim();
22         if (name.length() <= 0) throw new IllegalArgumentException("name ist leer.");
23         _name = name;
24         _normName = name.toLowerCase();
25         // Erzeugen der Singleton-Spezies der Spezifikation
26         _speciesSingleton = new SpeciesSingleton();
27         _speciesSingleton.setParent(this);
28         // Erzeugen der Spezies der Charakteristik der leeren Menge
29         _speciesOne = new SpeciesOne();
30         _speciesOne.setParent(this);
31     } // public Specification(String name) ...
32
33     /**
34      * Spezifikations-Kollektion, der die
35      * Spezifikation angehoert.
36      */
37     private SpecificationCollection _parent = null;
38
39     /**
40      * Gibt die Spezifikations-Kollektion zurueck,
41      * der die Spezifikation angehoert.
42      * @return Spezifikations-Kollektion, der die Spezifikation angehoert.
43      */
44
45     /**
46      * @see ISpecification#getSpecificationCollection()
47      */
48     public ISpecificationCollection getSpecificationCollection() {
49         return _parent;
50     } // public ISpecificationCollection getSpecificationCollection() ...
51
52     /**
53      * Setzt die Spezifikations-Kollektion, der die
54      * angegebene Kollektion angehoert.
55      * @param parent Spezifikations-Kollektion, der die
56      *   angegebene Kollektion angehoert.
57      */
58     protected void setParent(SpecificationCollection parent) {
59         _parent = parent;
60     } // protected void setParent(SpecificationCollection parent) ...
61
62     /**
63      * Name der Spezifikation.
64      */
65     private String _name = "";
66
67     /**
68      * Normierter Name der Spezifikation.
69      */
70     private String _normName = "";
71
72     /**
73      * Gibt den Namen der kominatorischen Spezifikation zurueck.
74      * @return Name der kombinatorischen Spezifikation.
75      */
76     public String getName() {
77         return _name;
78     } // public String getName() ...
79
```

```
80      /**
81       * Gibt den normierten Namen der kombinatorischen Spezifikation zurueck.
82       * @return Normierter Name der kombinatorischen Spezifikation.
83       */
84      protected String getNormName() {
85          return _normName;
86      } // protected String getNormName() ...
87
88      /**
89       * Liste der Spezies, die der Spezifikation angehoeren.
90       */
91      private Vector<NamedSpecies> _speciesList = new Vector<NamedSpecies>();
92
93      /**
94       * Gibt die Anzahl der Spezies, aus denen sich
95       * das kombinatorische System zusammensetzt,
96       * zurueck.
97       * @return Anzahl der Spezies, aus denen sich
98       *   das kombinatorisch System zusammensetzt.
99       */
100     public int getCount() {
101         return _speciesList.size();
102     } // public int getCount() ...
103
104     /**
105      * Gibt die Spezies mit dem angegebenen Index
106      * zurueck. Der Index muss im Bereich 0 .. getCount() - 1 liegen.
107      * @param index Index der gewuenschten Spezies.
108      * @return Spezies mit dem angegebenen Index.
109      * @throws IndexOutOfBoundsException Wird geworfen,
110      *   wenn der angegebene Index ausserhalb des gueltigen
111      *   Bereichs ist.
112      */
113     public INamedSpecies getSpeciesIndex(int index) {
114         if (index < 0 || index >= _speciesList.size()) throw new IndexOutOfBoundsException();
115         return _speciesList.elementAt(index);
116     } // public INamedSpecies getSpeciesIndex(int index) ...
117
118     /**
119      * Gibt die Spezies mit dem angegebenen Namen
120      * zurueck.
121      * @param name Name der Spezies, die zurueckgegeben werden soll.
122      * @return Spezies mit dem angegebenen Namen, falls vorhanden,
123      *   ansonsten null.
124      */
125     public INamedSpecies getSpecies(String name) {
126         if (name != null) {
127             String normName = name.trim().toLowerCase();
128             int l = 0;
129             int r = _speciesList.size() - 1;
130             while (l <= r) {
131                 int m = (l + r) / 2;
132                 NamedSpecies currentSpecies = _speciesList.elementAt(m);
133                 String currentName = currentSpecies.getNormName();
134                 int comparison = currentName.compareTo(normName);
135                 if (comparison < 0) {
136                     l = m + 1;
137                 } else if (comparison > 0) {
138                     r = m - 1;
139                 } else { // if (comparison == 0) ...
140                     return currentSpecies;
141                 } // if (comparison == 0) ...
142             } // while (l <= r) ...
143         } // if (name != null) ...
144         return null;
145     } // public INamedSpecies getSpecies(String name) ...
146
147     /**
148      * Fuegt die angegebene Spezies zur Collection hinzu.
149      * Der Parent der Spezies wird auf die aktuelle Instanz gesetzt.
150      * @param species Spezies, die hinzugefuegt werden soll.
151      * @return true, falls die Spezies hinzugefuegt werden konnte,
152      *   da noch keine zweite Spezies mit dem selben Namen existert,
153      *   ansonsten false.
154      * @throws IllegalArgumentException Wird geworfen, wenn die
155      *   angegebene Spezies null ist.
156      */
157     private boolean addSpecies(NamedSpecies species) {
158         if (species == null) throw new IllegalArgumentException("species = null.");
159         int l = 0;
```

```
160         int r = _speciesList.size() - 1;
161         String newName = species.getNormName();
162         while (l <= r) {
163             int m = (l + r) / 2;
164             NamedSpecies currentSpecies = _speciesList.elementAt(m);
165             String currentName = currentSpecies.getNormName();
166             int comparison = currentName.compareTo(newName);
167             if (comparison < 0) {
168                 l = m + 1;
169             } else if (comparison > 0) {
170                 r = m - 1;
171             } else { // if (comparison == 0) ...
172                 // Es gibt schon eine Spezies mit dem Namen.
173                 return false;
174             } // if (comparison == 0) ...
175         } // while (l <= r) ...
176         _speciesList.insertElementAt(species, l);
177         species.setParent(this);
178         return true;
179     } // private boolean addSpecies(NamedSpecies species) ...
180
181     /**
182      * Singleton-Spezies der Spezifikation.
183      */
184     private SpeciesSingleton _speciesSingleton;
185
186     /**
187      * Gibt die Singleton-Spezies zurueck.
188      * @return Singleton-Spezies.
189      */
190     public ISpeciesSingleton getSpeciesSingleton() {
191         return _speciesSingleton;
192     } // public ISpeciesSingleton getSingletonSpecies() ...
193
194     /**
195      * Spezies der Charakteristik der leeren Menge der Spezifikation.
196      */
197     private SpeciesOne _speciesOne;
198
199     /**
200      * Gibt die Spezies der Charakteristik der leeren Menge zurueck.
201      * @return Spezies der Charakteristik der leeren Menge.
202      */
203     public ISpeciesOne getSpeciesOne() {
204         return _speciesOne;
205     } // public ISpeciesOne getSpeciesOne() ...
206
207     /**
208      * Standard-Spezies.
209      */
210     private NamedSpecies _defaultSpecies = null;
211
212     /**
213      * Gibt die Standard-Spezies zurueck.
214      * @return Standard-Spezies.
215      */
216     public INamedSpecies getDefaultSpecies() {
217         return _defaultSpecies;
218     } // public INamedSpecies getDefaultSpecies() ...
219
220     /**
221      * LabelCollection der Spezifikation.
222      */
223     private LabelCollection _labelCollection = null;
224
225     /**
226      * Gibt die Label-Collection der Labels, die von
227      * der Spezifikation verwendet werden, zurueck.
228      * @return Label-Collection der Labels, die
229      *   von der Spezifikation verwendet werden.
230      */
231     public ILabelCollection getLabelCollection() {
232         return _labelCollection;
233     } // public ILabelCollection getLabelCollection() ...
234
235     /**
236      * Erzeugt aus dem uebergebenen Xml-Element eine
237      * kombinatorische Spezifikation.
238      * @param xmlElement Xml-Element, aus dem die Spezifikation erzeugt werden soll.
239      * @return Kombinatorische Spezifikation, die aus dem uebergebenen Xml-Element
```

```
240      *   erzeugt wurde.
241      * @throws SpecificationXmlLoadException Wird geworfen, wenn das uebergebene
242      *   Xml-Element die falsche Struktur besitzt.
243      * @throws IllegalArgumentException Wird geworfen, wenn das uebergebene
244      *   Xml-Element null ist.
245      */
246     public static Specification createFromXml(org.w3c.dom.Element xmlElement)
247       throws SpecificationXmlLoadException {
248         if (xmlElement == null) throw new IllegalArgumentException("xmlElement = null.");
249         // Namen der Spezifikation und Namen der Standard-Spezies laden
250         String nameSpecification = null;
251         String nameDefaultSpecies = null;
252         {
253             NamedNodeMap attributes = xmlElement.getAttributes();
254             if (attributes != null) {
255                 int length = attributes.getLength();
256                 for (int c = 0; c < length; c++) {
257                     Node currentAttribute = attributes.item(c);
258                     if (currentAttribute != null) {
259                         String namespaceURI = currentAttribute.getNamespaceURI();
260                         if (namespaceURI == null || SpecificationCollection.
                                SPECIFICATION_NAMESPACE.equals(namespaceURI)) {
261                             String attributeName = currentAttribute.getLocalName();
262                             if (attributeName != null) {
263                                 attributeName = attributeName.trim().toLowerCase();
264                                 if (attributeName.equals("name")) {
265                                     // Namen der Spezifikation laden
266                                     if (nameSpecification != null) {
267                                         throw new SpecificationXmlLoadException("Name der
                                                Spezifikation mehrmals angegeben.");
268                                     } // if (nameSpecification != null) ...
269                                     nameSpecification = currentAttribute.getNodeValue();
270                                     if (nameSpecification == null) {
271                                         nameSpecification = "";
272                                     } else { // if (nameSpecification != null) ...
273                                         nameSpecification = nameSpecification.trim();
274                                     } // if (nameSpecification != null) ...
275                                 } else if (attributeName.equals("defaultspecies")) {
276                                     // Namen der Standard-Spezies laden
277                                     if (nameDefaultSpecies != null) {
278                                         throw new SpecificationXmlLoadException("Standard-
                                                Spezies der Spezifikation mehrmals angegeben.");
279                                     } // if (nameDefaultSpecies != null) ...
280                                     nameDefaultSpecies = currentAttribute.getNodeValue();
281                                     if (nameDefaultSpecies == null) {
282                                         nameDefaultSpecies = "";
283                                     } else { // if (nameDefaultSpecies != null) ...
284                                         nameDefaultSpecies = nameDefaultSpecies.trim();
285                                     } // if (nameDefaultSpecies != null) ...
286                                 }
287                             } // if (attributeName != null) ...
288                         } // if (namespaceURI == null || SpecificationCollection.
                            SPECIFICATION_NAMESPACE.equals(namespaceURI)) ...
289                     } // if (currentAttribute != null) ...
290                 } // for (int c = 0; c < length; c++) ...
291             } // if (attributes != null) ...
292         }
293         if (nameSpecification == null || nameSpecification.length() <= 0) {
294             throw new SpecificationXmlLoadException("Kein Name fuer Spezifikation angegeben.");
295         } // if (nameSpecification == null || nameSpecification.length() <= 0) ...
296
297         // Spezifikation erzeugen
298         Specification result = new Specification(nameSpecification);
299         NodeList childNodes = xmlElement.getChildNodes();
300         if (childNodes != null) {
301             int length = childNodes.getLength();
302             for (int c = 0; c < length; c++) {
303                 Node currentChild = childNodes.item(c);
304                 if (currentChild != null && currentChild.getNodeType() == Node.ELEMENT_NODE &&
                        currentChild instanceof Element) {
305                     Element currentElement = (Element) currentChild;
306                     if (SpecificationCollection.SPECIFICATION_NAMESPACE.equals(currentElement.
                            getNamespaceURI())) {
307                         String currentElementName = currentElement.getLocalName();
308                         if (currentElementName == null) { currentElementName = ""; }
309                         currentElementName = currentElementName.trim().toLowerCase();
310                         if (currentElementName.equals(LabelCollection.XML_ELEMENT_NAME)) {
311                             // Label-Collection laden.
312                             if (result._labelCollection != null) {
```

```
313                                 throw new SpecificationXmlLoadException("Mehrere Label-
                                        Kollektionen angegeben.");
314                             }
315                             LabelCollection newLabelCollection = LabelCollection.createFromXml(
                                    currentElement);
316                             result._labelCollection = newLabelCollection;
317                             newLabelCollection.setParent(result);
318                         } else if (NamedSpecies.isKnownNamedSpeciesElement(currentElementName))
                                {
319                             // Benannte Spezies laden.
320                             NamedSpecies newSpecies = NamedSpecies.createFromXml(currentElement
                                    );
321                             if (!result.addSpecies(newSpecies)) {
322                                 throw new SpecificationXmlLoadException("Mehrere Spezies mit
                                        dem selben Namen angegeben.");
323                             } // if (!result.addSpecies(newSpecies)) ...
324                         } // if (NamedSpecies.isKnownNamedSpeciesElement(currentElementName))
                                ...
325                     } // if (SpecificationCollection.SPECIFICATION_NAMESPACE.equals(
                            currentElement.getNamespaceURI())) ...
326                 } // if (currentChild != null && currentChild.getNodeType() == Node.
                        ELEMENT_NODE && currentChild instanceof Element) ...
327             } // for (int c = 0; c < length; c++) ...
328         } // if (childNodes != null) ...
329
330         // Falls keine Label-Collection angegeben wurde, dann eine leere Collection erzeugen.
331         if (result._labelCollection == null) {
332             result._labelCollection = new LabelCollection();
333             result._labelCollection.setParent(result);
334         } // if (result._labelCollection == null) ...
335
336         // Standard-Spezies zuordnen, falls angegeben
337         if (nameDefaultSpecies != null) {
338             INamedSpecies species = result.getSpecies(nameDefaultSpecies);
339             if (species instanceof NamedSpecies) {
340                 result._defaultSpecies = (NamedSpecies) species;
341             } else { // if (!(species instanceof NamedSpecies)) ...
342                 throw new SpecificationXmlLoadException("Standard-Spezies \"" +
                            nameDefaultSpecies + "\" nicht gefunden.");
343             } // if (!(species instanceof NamedSpecies)) ...
344         } else { // if (nameDefaultSpecies == null) ...
345             result._defaultSpecies = null;
346         } // if (nameDefaultSpecies == null) ...
347
348         // Indizes der benannten Spezies setzen
349         {
350             int length = result._speciesList.size();
351             for (int c = 0; c < length; c++) {
352                 result._speciesList.elementAt(c).setIndex(c);
353             } // for (int c = 0; c < length; c++) ...
354         }
355
356         return result;
357     } // public static Specification createFromXml(org.w3c.dom.Element xmlElement) ...
358
359     /**
360      * Beendet den Xml-Ladevorgang.
361      * @throws at.techmath.boltzmann.specification.SpecificationXmlLoadException Wird geworfen,
362      *   wenn ein unbekanntes Objekt referenziert wird.
363      */
364     protected void finishXmlLoad() throws SpecificationXmlLoadException {
365         for (NamedSpecies namedSpecies : _speciesList) {
366             namedSpecies.finishXmlLoad();
367         } // for (NamedSpecies namedSpecies : _speciesList) ...
368     } // protected void finishXmlLoad() throws SpecificationXmlLoadException ...
369
370     /**
371      * Gibt eine String-Repraesentation der Instanz zurueck.
372      * @return String-Repraesentation der Instanz.
373      */
374     @Override
375     public String toString() {
376         StringBuilder sb = new StringBuilder();
377         sb.append("  Spezifikation \"");
378         sb.append(getName());
379         sb.append("\"");
380         // Standardspezies ausgeben, falls gesetzt
381         if (_defaultSpecies != null) {
382             sb.append(" (Standard: \"");
383             sb.append(_defaultSpecies.getName());
```

```
384             sb.append("\")");
385         } // if (_defaultSpecies != null) ...
386         sb.append("\n");
387         // Spezies ausgeben
388         sb.append("    Spezies\n");
389         for (NamedSpecies namedSpecies : _speciesList) {
390             sb.append(namedSpecies.toString());
391         } // for (NamedSpecies namedSpecies : _speciesList) ...
392         // Label-Collection ausgeben
393         if (_labelCollection != null) {
394             sb.append(_labelCollection.toString());
395         } // if (_labelCollection != null) ...
396         return sb.toString();
397     } // public String toString() ...
398 } // class Specification implements ISpecification ...
```

Listing 165: Klasse *at.techmath.boltzmann.specification.SpecificationCollection*

```
1  package at.techmath.boltzmann.specification;
2
3  import java.util.Vector;
4  import org.w3c.dom.*;
5
6  /**
7   * Standard-Implementierung des ISpecificationCollection-Interfaces.
8   * @author Stefan Schnabl (e0226245)
9   */
10 class SpecificationCollection implements ISpecificationCollection {
11     /**
12      * Namespace der Spezifikation.
13      */
14     public static final String SPECIFICATION_NAMESPACE = "http://techmath.at/boltzmann/
           specification/2014/11/06";
15
16     /**
17      * Liste der Spezifikationen, nach normiertem Namen
18      * aufsteigend sortiert.
19      */
20     private Vector<Specification> _specificationList = new Vector<Specification>();
21
22     /**
23      * Gibt die Anzahl der kombinatorischen Spezifikationen
24      * zurueck.
25      * @return Anzahl der kombinatorischen Spezifikationen.
26      */
27     public int getCount() {
28         return _specificationList.size();
29     } // public int getCount() ...
30
31     /**
32      * Gibt die Spezifikation mit dem angegebenen Index
33      * zurueck. Der Index muss zwischen 0 und getCount() - 1 liegen.
34      * @param index Index der Spezifikation, die ermittelt werden soll.
35      * @return Spezifikation mit dem angegebenen Index.
36      * @throws IndexOutOfBoundsException Der angegebene Index liegt ausserhalb
37      *   des gueltigen Bereichs.
38      */
39     public ISpecification getSpecificationIndex(int index) {
40         if (index < 0 || index >= _specificationList.size()) {
41             throw new IndexOutOfBoundsException("Der Index liegt ausserhalb des gueltigen
                   Bereichs.");
42         } // if (index < 0 || index >= _specificationList.size()) ...
43         return _specificationList.elementAt(index);
44     } // public ISpecification getSpecificationIndex(int index) ...
45
46     /**
47      * Gibt die Spezifikation mit dem angegebenen Namen zurueck.
48      * @param name Name der gewuenschten Spezifikation.
49      * @return Spezifikation mit dem angegebenen Namen,
50      *   falls vorhanden, ansonsten null.
51      */
52     public ISpecification getSpecification(String name) {
53         String normName;
54         if (name == null) {
55             normName = "";
56         } else { // if (name != null) ...
57             normName = name.trim().toLowerCase();
58         } // if (name != null) ...
59         int l = 0;
60         int r = _specificationList.size() - 1;
61         while (l <= r) {
62             int m = (l + r) / 2;
63             Specification currentSpecification = _specificationList.elementAt(m);
64             String currentName = currentSpecification.getNormName();
65             int comparison = currentName.compareTo(normName);
66             if (comparison < 0) {
67                 l = m + 1;
68             } else if (comparison > 0) {
69                 r = m - 1;
70             } else { // if (comparison == 0) ...
71                 return currentSpecification;
72             } // if (comparison == 0) ...
73         } // while (l <= r) ...
74         return null;
75     } // public ISpecification getSpecification(String name) ...
76
77     /**
```

```
78       * Fuegt die angegebene Spezifikation hinzu.
79       * Falls die Spezifikation hinzugefuegt werden kann,
80       * dann wird deren Parent auf die aktuelle Instanz gesetzt.
81       * @param specification Spezifikation, die hinzugefuegt werden soll.
82       * @return true, falls die angegebene Spezifikation hinzugefuegt
83       *   werden konnte, falls also noch keine Spezifikation mit dem
84       *   Namen der angegebenen Spezifikation in der Collection vorhanden war,
85       *   ansonsten false.
86       * @throws IllegalArgumentException Wird geworfen, wenn die
87       *   angegebene Spezifikation null ist.
88       */
89      private boolean addSpecification(Specification specification) {
90          if (specification == null) throw new IllegalArgumentException("specification = null.");
91          int l = 0;
92          int r = _specificationList.size() - 1;
93          String newName = specification.getNormName();
94          while (l <= r) {
95              int m = (l + r) / 2;
96              Specification currentSpecification = _specificationList.elementAt(m);
97              String currentName = currentSpecification.getNormName();
98              int comparison = currentName.compareTo(newName);
99              if (comparison < 0) {
100                 l = m + 1;
101             } else if (comparison > 0) {
102                 r = m - 1;
103             } else { // if (comparison == 0) ...
104                 // Es existiert in der Collection schon
105                 // eine Spezifikation mit dem angegebenen Namen.
106                 return false;
107             } // if (comparison == 0) ...
108         } // while (l <= r) ...
109         _specificationList.insertElementAt(specification, l);
110         specification.setParent(this);
111         return true;
112     } // private boolean addSpecification(Specification specification) ...
113
114     /**
115      * Standard-Spezifikation.
116      */
117     private Specification _defaultSpecification = null;
118
119     /**
120      * Gibt die Standard-Spezifikation zurueck.
121      * @return Standard-Spezifikation.
122      */
123     public ISpecification getDefaultSpecification() {
124         return _defaultSpecification;
125     } // public ISpecification getDefaultSpecification() ...
126
127     /**
128      * Erzeugt aus dem angegebenen Xml-Element eine
129      * neue SpecificationCollection-Instanz.
130      * @param xmlElement Xml-Element, aus dem die neue Instanz erzeugt werden soll.
131      * @return SpecificationCollection-Instanz, die aus dem angegebenen
132      *   Xml-Element erzeugt wurde.
133      * @throws at.techmath.boltzmann.SpecificationXmlLoadException Wird geworfen, wenn
134      *   das uebergebene Xml-Element nicht die erwartete Struktur besitzt.
135      * @throws IllegalArgumentException Wird geworfen, wenn das angegebene
136      *   Xml-Element null ist.
137      */
138     public static SpecificationCollection createFromXml(org.w3c.dom.Element xmlElement)
139       throws SpecificationXmlLoadException {
140         if (xmlElement == null) throw new IllegalArgumentException("xmlElement = null.");
141         SpecificationCollection result = new SpecificationCollection();
142
143         // Namen der Standard-Spezifikation aus den Attributen laden
144         String nameDefaultSpecification = null;
145         {
146             NamedNodeMap attributes = xmlElement.getAttributes();
147             if (attributes != null) {
148                 int length = attributes.getLength();
149                 for (int c = 0; c < length; c++) {
150                     Node currentAttribute = attributes.item(c);
151                     if (currentAttribute != null) {
152                         String namespaceURI = currentAttribute.getNamespaceURI();
153                         if (namespaceURI == null || SpecificationCollection.
                                SPECIFICATION_NAMESPACE.equals(namespaceURI)) {
154                             String attributeName = currentAttribute.getLocalName();
155                             if (attributeName == null) {
156                                 attributeName = "";
```

```
157                             } else { // if (attributeName != null) ...
158                                 attributeName = attributeName.trim().toLowerCase();
159                             } // if (attributeName != null) ...
160                             if (attributeName.equals("defaultspecification")) {
161                                 if (nameDefaultSpecification != null) {
162                                     throw new SpecificationXmlLoadException("Standard-
                                            Spezifikation mehrmals angegeben.");
163                                 } // if (nameDefaultSpecification != null) ...
164                                 nameDefaultSpecification = currentAttribute.getNodeValue();
165                                 if (nameDefaultSpecification == null) {
166                                     nameDefaultSpecification = "";
167                                 } else {
168                                     nameDefaultSpecification = nameDefaultSpecification.trim();
169                                 }
170                             } // if (attributeName.equals("defaultspecification")) ...
171                         } // if (namespaceURI == null || SpecificationCollection.
                                SPECIFICATION_NAMESPACE.equals(namespaceURI)) ...
172                     } // if (currentAttribute != null) ...
173                 } // for (int c = 0; c < length; c++) ...
174             } // if (attributes != null) ...
175         }
176
177         // Spezifikationen laden
178         {
179             NodeList childNodes = xmlElement.getChildNodes();
180             if (childNodes != null) {
181                 int length = childNodes.getLength();
182                 for (int c = 0; c < length; c++) {
183                     Node currentNode = childNodes.item(c);
184                     if (currentNode != null && currentNode.getNodeType() == Node.ELEMENT_NODE
                            && currentNode instanceof Element) {
185                         Element currentElement = (Element) currentNode;
186                         if (SpecificationCollection.SPECIFICATION_NAMESPACE.equals(
                                currentElement.getNamespaceURI())) {
187                             String currentElementName = currentElement.getLocalName();
188                             if (currentElementName == null) {
189                                 currentElementName = "";
190                             } else { // if (currentElementName != null) ...
191                                 currentElementName = currentElementName.trim().toLowerCase();
192                             } //if (currentElementName != null) ...
193                             if (currentElementName.equals("specification")) {
194                                 Specification newSpecification = Specification.createFromXml(
                                        currentElement);
195                                 if (!result.addSpecification(newSpecification)) {
196                                     throw new SpecificationXmlLoadException("Mehrere
                                            Spezifikationen mit dem selben Namen angegeben.");
197                                 }
198                             }
199                         } // if (SpecificationCollection.SPECIFICATION_NAMESPACE.equals(
                                currentElement.getNamespaceURI())) ...
200                     } // if (currentNode != null && currentNode.getNodeType() == Node.
                            ELEMENT_NODE && currentNode instanceof Element) ...
201                 } // for (int c = 0; c < length; c++) ...
202             } // if (childNodes != null) ...
203         }
204
205         // Standard-Spezifikation setzen
206         if (nameDefaultSpecification != null) {
207             Specification specification = (Specification) result.getSpecification(
                    nameDefaultSpecification);
208             if (specification != null) {
209                 result._defaultSpecification = specification;
210             } else { // if (specification == null) ...
211                 throw new SpecificationXmlLoadException("Angegebene Standard-Spezifikation
                        nicht gefunden.");
212             } // if (specification == null) ...
213         } // if (nameDefaultSpecification != null) ...
214
215         // Falls nur eine Spezifikation vorhanden ist, und
216         // keine explizite Standard-Spezifikation angegeben wurde,
217         // dann wird diese einzige als Standard-Spezifikation festgelegt.
218         if (result._defaultSpecification == null && result.getCount() == 1) {
219             result._defaultSpecification = (Specification) result.getSpecificationIndex(0);
220         } // if (result._defaultSpecification == null && result.getCount() == 1) ...
221
222         for (Specification specification : result._specificationList) {
223             specification.finishXmlLoad();
224         } // for (Specification specification : result._specificationList) ...
225
226         return result;
```

```
227     } // public static SpecificationCollection createFromXml(org.w3c.dom.Element xmlElement)
            ...
228
229     /**
230      * Gibt eine String-Repraesentation der Instanz zurueck.
231      * @return String-Repraesentation der Instanz.
232      */
233     @Override
234     public String toString() {
235         StringBuilder sb = new StringBuilder();
236         sb.append("Spezifikations-Kollektion");
237         if (_defaultSpecification != null) {
238             sb.append(" (Standard: \"");
239             sb.append(_defaultSpecification.getName());
240             sb.append("\")");
241         } // if (_defaultSpecification != null) ...
242         sb.append("\n");
243         int count = getCount();
244         for (int c = 0; c < count; c++) {
245             ISpecification currentSpecification = getSpecificationIndex(c);
246             sb.append(currentSpecification.toString());
247             sb.append("\n");
248         } // for (int c = 0; c < count; c++) ...
249         return sb.toString();
250     } // public String toString() ...
251 } // class SpecificationCollection implements ISpecificationCollection ...
```

Listing 166: Klasse *at.techmath.boltzmann.specification.SpecificationCollectionXmlLoader*

```
1  package at.techmath.boltzmann.specification;
2
3  import javax.xml.parsers.*;
4  import java.io.IOException;
5  import java.io.FileNotFoundException;
6  import java.io.InputStream;
7  import java.io.FileInputStream;
8  import org.xml.sax.SAXException;
9  import org.w3c.dom.Document;
10 import org.w3c.dom.Element;
11 import org.w3c.dom.Node;
12 import org.w3c.dom.NodeList;
13
14 /**
15  * Klasse, die eine Kollektion kombinatorischer Spezifikationen aus einer
16  * Xml-Datei laden kann.
17  * @author Stefan Schnabl (e0226245)
18  */
19 public class SpecificationCollectionXmlLoader {
20     /**
21      * Laedt eine Sammlung von kombinatorischen Spezifikationen
22      * aus der Xml-Datei mit dem angegebenen Pfad.
23      * @param path Pfad der Xml-Datei, die geladen werden soll.
24      * @return Sammlung von kombinatorischen Spezifikationen,
25      *   die aus der angegebenen Xml-Datei geladen wurde.
26      * @throws IllegalArgumentException Wird geworfen, wenn der angegebene
27      *   Pfad null ist.
28      */
29     public ISpecificationCollection loadFromFile(String path)
30       throws SpecificationXmlLoadException {
31         if (path == null) throw new IllegalArgumentException("path = null.");
32         FileInputStream fileInputStream = null;
33         try {
34             try {
35                 fileInputStream = new FileInputStream(path);
36             } catch (FileNotFoundException fileNotFoundException) {
37                 throw new SpecificationXmlLoadException("Datei '" + path + "' nicht gefunden.",
                        fileNotFoundException);
38             }
39             return loadFromStream(fileInputStream);
40         } finally {
41             if (fileInputStream != null) {
42                 try { fileInputStream.close(); } catch (IOException ioException) { }
43             } // if (fileInputStream != null) ...
44         }
45     } // public ISpecificationCollection loadFromFile(String path) ...
46
47     /**
48      * Laedt eine Sammlung von kombinatorischen Spezifikationen
49      * aus dem ubergebenen Stream, der die Spezifikation im Xml-Format liefert.
50      * @param stream Stream, der die Spezifikation im Xml-Format liefert.
51      * @return Sammlung von kombinatorischen Spezifikationen,
52      *   die aus dem angegebenen Xml-Stream geladen wurden.
53      * @throws IllegalArgumentException Wird geworfen, wenn
54      *   der angegebene Stream null ist.
55      * @throws at.techmath.boltzmann.specification.SpecificationXmlLoadException Wird geworfen,
56      *   wenn, beim Laden der Xml-Spezifikation ein Fehler aufgetreten ist.
57      */
58     public ISpecificationCollection loadFromStream(InputStream stream)
59       throws SpecificationXmlLoadException {
60         if (stream == null) throw new IllegalArgumentException("stream = null.");
61         DocumentBuilderFactory documentBuilderFactory;
62         try {
63             documentBuilderFactory = DocumentBuilderFactory.newInstance();
64             documentBuilderFactory.setNamespaceAware(true);
65         } catch (FactoryConfigurationError factoryConfigurationError) {
66             throw new SpecificationXmlLoadException("Xml-DocumentBuilderFactory falsch
                    konfiguriert.", factoryConfigurationError);
67         }
68         DocumentBuilder documentBuilder;
69         try {
70             documentBuilder = documentBuilderFactory.newDocumentBuilder();
71         } catch (ParserConfigurationException parserConfigurationException) {
72             throw new SpecificationXmlLoadException("Xml-Parser falsch konfiguriert.",
                    parserConfigurationException);
73         }
74         Document document;
75         try {
76             document = documentBuilder.parse(stream);
```

```
77          } catch (IOException ioException) {
78              String message = ioException.getMessage(); if (message == null) { message = "."; }
                    else { message = ": " + ioException.getMessage(); }
79              throw new SpecificationXmlLoadException("I/O-Fehler beim Laden der Xml-
                    Spezifikation" + message, ioException);
80          } catch (SAXException saxException) {
81              String message = saxException.getMessage(); if (message == null) { message = "."; }
                     else { message = ": " + saxException.getMessage(); }
82              throw new SpecificationXmlLoadException("Xml-Parser-Fehler beim Laden der Xml-
                    Spezifikation" + message, saxException);
83          }
84          SpecificationCollection result = null;
85          {
86              NodeList childNodes = document.getChildNodes();
87              if (childNodes != null) {
88                  int length = childNodes.getLength();
89                  for (int c = 0; c < length; c++) {
90                      Node currentNode = childNodes.item(c);
91                      if (currentNode != null && currentNode.getNodeType() == Node.ELEMENT_NODE
                             && currentNode instanceof Element) {
92                          Element currentElement = (Element) currentNode;
93                          if (SpecificationCollection.SPECIFICATION_NAMESPACE.equals(
                                 currentElement.getNamespaceURI())) {
94                              String currentElementName = currentElement.getNodeName();
95                              if (currentElementName == null) {
96                                  currentElementName = "";
97                              } else { // if (elementName != null) ...
98                                  currentElementName = currentElementName.trim().toLowerCase();
99                              } // if (elementName != null) ...
100                             if (currentElementName.equals("specifications")) {
101                                 if (result != null) {
102                                     throw new SpecificationXmlLoadException("Spezifikations-
                                            Kolletion mehrmals angegeben.");
103                                 } // if (result != null) ...
104                                 result = SpecificationCollection.createFromXml(currentElement);
105                             }
106                         } // if (SpecificationCollection.SPECIFICATION_NAMESPACE.equals(
                                currentElement.getNamespaceURI())) ...
107                     } // if (currentNode != null && currentNode.getNodeType() == Node.
                            ELEMENT_NODE && currentNode instanceof Element) ...
108                 } // for (int c = 0; c < length; c++) ...
109             } // if (childNodes != null) ...
110         }
111         if (result == null) {
112             throw new SpecificationXmlLoadException("Keine Spezifikations-Kollektion gefunden."
                    );
113         } // if (result == null) ...
114         return result;
115     } // public ISpecificationCollection loadFromStream(InputStream stream) ...
116 } // public class SpecificationCollectionXmlLoader ...
```

Listing 167: Klasse *at.techmath.boltzmann.specification.SpecificationXmlLoadException*

```
1  package at.techmath.boltzmann.specification;
2
3  /**
4   * Exception, die geworfen wird, wenn beim Laden
5   * einer kombinatorischen Spezifikation aus einer Xml-Datei
6   * ein Fehler aufgetreten ist.
7   * @author Stefan Schnabl (e0226245)
8   */
9  public class SpecificationXmlLoadException extends Exception {
10     private static final long serialVersionUID = 4731052296460323665L;
11     public SpecificationXmlLoadException() { super(); }
12     public SpecificationXmlLoadException(String message) { super(message); }
13     public SpecificationXmlLoadException(String message, Throwable cause) { super(message,
           cause); }
14     public SpecificationXmlLoadException(Throwable cause) { super(cause); }
15 } // public class SpecificationXmlLoadException extends Exception ...
```

Listing 168: Klasse *at.techmath.boltzmann.specification.StringLabel*

```
1  package at.techmath.boltzmann.specification;
2
3  import org.w3c.dom.Node;
4  import org.w3c.dom.NodeList;
5
6  /**
7   * Standard-Implementierung des IStringLabel-Interfaces.
8   * @author Stefan Schnabl (e0226245)
9   */
10 class StringLabel extends Label implements IStringLabel {
11     /**
12      * Name des Xml-Elements.
13      */
14     public static final String XML_ELEMENT_NAME = "textlabel";
15
16     /**
17      * Erzeugt eine neue Instanz.
18      * @param name Name des Labels.
19      * @param text Textinhalt des Labels.
20      */
21     public StringLabel(String name, String text) {
22         super(name);
23         _text = text != null ? text : "";
24     } // public StringLabel(ILabelCollection parent, String name, String text) ...
25
26     /**
27      * Text-Inhalt des Labels.
28      */
29     private String _text = null;
30
31     /**
32      * Gibt den Textinhalt des Labels zurueck.
33      * @return Textinhalt des Labels.
34      */
35     public String getText() {
36         return _text;
37     } // public String getText() ...
38
39     /**
40      * Erzeugt aus dem gegebenen Xml-Element eine StringLabel-Instanz.
41      * @param name Name des Labels.
42      * @param xmlElement Xml-Element, aus dem das Label erzeugt werden soll.
43      * @return StringLabel-Instanz, die aus dem uebergebenen Xml-Element erzeugt wurde.
44      */
45     protected static StringLabel createFromXml(String name, org.w3c.dom.Element xmlElement) {
46         if (xmlElement == null) throw new IllegalArgumentException("xmlElement = null");
47         // Text-Inhalt ermitteln.
48         String textContent = "";
49         NodeList childs = xmlElement.getChildNodes();
50         if (childs != null) {
51             StringBuilder sb = new StringBuilder();
52             int length = childs.getLength();
53             for (int c = 0; c < length; c++) {
54                 Node currentNode = childs.item(c);
55                 if (currentNode != null && currentNode.getNodeType() == Node.TEXT_NODE) {
56                     String currentText = currentNode.getNodeValue();
57                     if (currentText != null) {
58                         sb.append(currentText);
59                     } // if (currentText != null) ...
60                 } // if (currentNode != null && currentNode.getNodeType() == Node.TEXT_NODE)
                       ...
61             } // for (int c = 0; c < length; c++) ...
62             textContent = sb.toString().trim();
63         } // if (childs != null) ...
64         return new StringLabel(name, textContent);
65     } // protected static StringLabel createFromXml(String name, org.w3c.dom.Element xmlElement
           ) ...
66
67     /**
68      * Gibt die String-Repraesentation der Instanz zurueck.
69      * @return String-Repraesentation der Instanz.
70      */
71     @Override
72     public String toString() {
73         StringBuilder sb = new StringBuilder();
74         sb.append("Text-Label (\"");
75         sb.append(getName());
76         sb.append("\", \"");
77         sb.append(getText());
```

```
78          sb.append("\")");
79          return sb.toString();
80      } // public String toString() ...
81  } // class StringLabel extends Label implements IStringLabel ...
```

Literatur

[1] David Aldous. On the markov chain simulation method for uniform combinatorial distributions and simulated annealing. *Probability in the Engineering and Informational Sciences*, 1(01):33–46, 1987.

[2] François Bergeron. *Combinatorial species and tree-like structures*, volume 67. Cambridge University Press, 1998.

[3] Manuel Bodirsky, Éric Fusy, Mihyun Kang, and Stefan Vigerske. Boltzmann samplers, pólya theory, and cycle-pointing. *SIAM Journal on Computing*, 40(3):721–769, 2011.

[4] Philippe Duchon, Philippe Flajolet, Guy Louchard, and Gilles Schaeffer. Random sampling from boltzmann principles. In *Automata, languages and programming*, pages 501–513. Springer, 2002.

[5] Philippe Duchon, Philippe Flajolet, Guy Louchard, and Gilles Schaeffer. Boltzmann samplers for the random generation of combinatorial structures. *Combinatorics, Probability and Computing*, 13(4–5):577–625, 2004. Special issue on Analysis of Algorithms.

[6] Philippe Flajolet, P.Zimmerman, and B. van Cutsem. A calculus for the random generation of labelled combinatorial structures. *Theoretical Computer Science*, 132(1–2):1–35, 1994.

[7] Philippe Flajolet and Robert Sedgewick. *Analytic combinatorics*. Cambridge University Press, 2009.

[8] Thomas W Hungerford. Algebra, volume 73 of graduate texts in mathematics, 1980.

[9] André Joyal. Une théorie combinatoire des séries formelles. *Advances in mathematics*, 42(1):1–82, 1981.

[10] AW Kemp. Efficient generation of logarithmically distributed pseudo-random variables. *Applied Statistiss*, pages 249–253, 1981.

[11] Norbert Kusolitsch. *Maß-und Wahrscheinlichkeitstheorie: Eine Einführung*. Springer-Verlag, 2014.

[12] Serge Lang. Algebra revised third edition. *GRADUATE TEXTS IN MATHEMATICS-NEW YORK-*, 2002.

[13] George Marsaglia. Random number generators. *Journal of Modern Applied Statistical Methods*, 2(1):2–13, 2003.

[14] Carine Pivoteau, Bruno Salvy, and Michèle Soria. Algorithms for combinatorial structures: Well-founded systems and newton iterations. *Journal of Combinatorial Theory, Series A*, 119(8):1711 – 1773, 2012. Available at http://fr.arxiv.org/abs/1109.2688.