Die approbierte Originalversion dieser Diplom-/Masterarbeit ist an der Hauptbibliothek der Technischen Universität Wien aufgestellt (http://www.ub.tuwien.ac.at).

The approved original version of this diploma or master thesis is available at the main library of the Vienna University of Technology (http://www.ub.tuwien.ac.at/englweb/).

TECHNISCHE UNIVERSITÄT WIEN

VIENNA UNIVERSITY OF TECHNOLOGY

M A G I S T E R A R B E I T

Wissensrepräsentation in RHEUMexpert/Web mit Lua

ausgeführt am

Institut für Medizinische Experten- und Wissensbasierte Systeme
der Besonderen Einrichtung für Medizinische Statistik und Informatik
der Medizinischen Universität Wien

Begutachter:
Univ.-Prof. Dipl.-Ing. Dr. techn. Klaus-Peter Adlassnig

Betreuer:
Univ.-Ass. Dipl.-Math. Dr. phil. Rudolf Seising
Dipl.-Ing. Reinhard Pitsch

Ersteller:
Markus Wissekal Bakk. techn.
Krallgasse 3; 1220 Wien
Mat.-Nr.: 9925122

Wien, am 28. September 2007
Datum

Unterschrift

Danksagung

Mein Dank gilt dem Institut für Medizinische Experten- und Wissensbasierte Systeme der Medizinischen Universität Wien, welches mir die Möglichkeit gab, an diesem Thema zu arbeiten. Im Speziellen bedanke ich mich bei meinen Betreuern Dipl.-Math. Dr. Rudolf Seising und DI Reinhard Pitsch, die mir beide mit Rat und Tat zur Verfügung standen, mir aber auch den benötigten Spielraum liessen, um meine Arbeit zu verfassen. Ich möchte mich bei DI Andrea Rappelsberger für die nützlichen Tipps und Anregungen bei der Korrektur meiner Diplomarbeit danken. DI Udo Kronberger verdient meinen Dank für die gute Zusammenarbeit bei unseren beiden Arbeiten.

Ich möchte mich bei Alfred Hirtl vom Bundesrechenzentrum bedanken, da er mir in den letzten Jahren viele Freiheiten liess, und mich bei meiner Diplomarbeit und meinem Auslandsaufenthalt in Schweden unterstützte.

Ein ganz besonderer Dank gebührt meinen Eltern, für deren Zeit, Liebe und der Möglichkeit mein Studium zu vollenden.

Zuletzt möchte ich mich bei Elena für ihre Geduld beim Zuhören, ihre Aufmunterungen und für vieles mehr bedanken.

Zusammenfassung

Im Zuge dieser wissenschaftlichen Arbeit wird die Planung und Entwicklung der Wissensverarbeitung von RHEUMexpert/Web durchgeführt. Hierdurch wird ein System, welches auf Regeln und Sicherheitsfaktoren basiert, erzeugt, welches in einer eigenen Grammatik abgearbeitet wird. Diese baut auf der Sprache Lua auf, einer äusserst flexiblen Skriptsprache, die in einer kompakten virtuellen Maschine abgearbeitet wird. Die neuen Regeln werden in den bestehenden RHEUMexpert/Web-Server eingebunden.

RHEUMexpert/Web ist ein medizinisches Expertensystem zur Erstellung von Verdachtsdiagnosen in der Rheumatologie. Frühere Systemversionen von RHEUMexpert hatten keine Möglichkeit die Wissensbasis anzupassen, da diese direkt im Programmcode realisiert war. In Folge dessen wurde im Zuge der Überarbeitung der Grammatik für RHEUMexpert/Web eine Auftrennung von Programmcode und Wissensverarbeitung vollzogen.

Die vorliegende Arbeit beinhaltet eine Einführung in die wichtigsten Themengebiete und *Materialien*, die zum Verständnis erforderlich sind. Diese sind eine Einführung in die Künstliche Intelligenz mit Überleitungen zu den Bereichen der Expertensysteme, der Sicherheitsfaktoren und deren Anwendung in der Inferenzmaschine, sowie ein Vergleich zwischen kompilierten und interpretierten Sprachen. Weiters werden Eigenschaften von Logiken, Kostenfunktionen, eine Einführung in Grammatiken, die verwendete Software und ein Überblick von Arbeiten zum Thema RHEUMexpert gegeben.

Im Weiteren erfolgt eine Beschreibung des praktischen Teils und der verwendeten *Methoden*, wodurch die Verbesserungen an der Wissensrepräsentation sichtbar werden. Diese unterteilt sich in die formale Definition der neuen Sprache, die Umwandlung der alten Regeln in die neue Form, die erstellten Arithmetik- und Logikoperatoren, die eigentlichen Inferenz und deren Einbindung in den RHEUMexpert/Web-Server.

In der abschließenden Diskussion werden die *Resultate* der vorliegenden Arbeit präsentiert und ein *Ausblick* für Weiterentwicklungen angeführt, um Hinweise für weitere Arbeiten an einer zentralen Stelle anzubieten.

Abstract

This thesis describes the design and implementation of the knowledge processing method in RHEUMexpert/Web. The inference system is handled in its own grammar that rests upon rule sets and certainty factors. The basis of its grammar is a sub set of Lua, an extremely flexible scripting language processed in a compact virtual machine. All steps of the created inference were entirely separated from the servers' source code.

RHEUMexpert/Web is a medical expert system that creates tentative diagnoses (diagnosis upon suspicion) in rheumatology. Previous versions of RHEUMexpert did not provide the possibility to adapt or improve the knowledge base automatically or via any kind of knowledge base editor as it was hard-coded within the source code. Here, an inference engine of RHEUMexpert/Web was recreated to detach the systems knowledge processing from its source code.

In terms of understanding, the preface discusses important topics and materials such as: an introduction to artificial intelligence in connection with medical expert systems, certainty factors and their usage in the created inference engine as well as a comparison of compiled and interpreted programming languages. Furthermore, the thesis illustrates different aspects of logic, cost functions, an introduction to grammars, the used software and an overview of other related works regarding RHEUMexpert.

The body of the thesis contains methods of the programming and analytical part in which new approaches to knowledge representation in RHEUMexpert can be seen. It consists of the new grammars formal definition, the conversion of old rules to this grammar, created arithmetical and logical operators, all steps of the inference and a description on how to embed them in the server.

The discussion summarizes the findings of the thesis. The results consist of the evaluating scientist's opportunity to adjust the rules of inference or values in the certainty factor matrix as well as the possibility to decide upon the method of choosing possible diagnoses during the runtime of the server. Finally an overview of possible future developments is given.

Inhaltsverzeichnis

Zusammenfassung **ii**

Abstract **iii**

Inhaltsverzeichnis **iv**

Abbildungsverzeichnis **v**

Programmlistings **vi**

1 Einleitung **1**
- 1.1 Themengebiet . . . 1
- 1.2 Gliederung . . . 1
- 1.3 Motivation . . . 3
- 1.4 Klinische Grundlagen . . . 4
- 1.5 Ziel von RHEUMexpert/Web . . . 5
- 1.6 Ziel der vorliegenden Arbeit . . . 7

2 Material / Verwandte Themengebiete **8**
- 2.1 Künstliche Intelligenz . . . 8
  - 2.1.1 Intelligente Systeme . . . 9
  - 2.1.2 Wissensbasierte Systeme . . . 10
  - 2.1.3 Expertensysteme . . . 11
  - 2.1.4 Erfolgreiche medizinische Expertensysteme . . . 12
  - 2.1.5 Arbeitsweise . . . 14
- 2.2 Sicherheitsfaktoren . . . 15
  - 2.2.1 Definition . . . 16
  - 2.2.2 Measures of (Dis-)Belief . . . 16
- 2.3 Inferenz . . . 16
- 2.4 Programmiersprachen . . . 18
  - 2.4.1 Kompilierte Sprachen . . . 18
  - 2.4.2 Interpretierte Sprachen . . . 19
  - 2.4.3 Virtuelle Maschinen . . . 20
  - 2.4.4 Lexen und Parsen . . . 21
- 2.5 Aussagenlogik . . . 22

2.5.1 Zweiwertige Logik 23
2.5.2 Dreiwertige Logik 23
2.5.3 Fuzzy-Logik 24
2.6 Landau-Symbole / Kostenfunktionen 25
2.7 Formale Grammatiken 26
2.7.1 (E)BNF 27
2.7.2 XML 29
2.7.3 EBNF der alten Wissensbasis 31
2.8 Softwarepakete 34
2.8.1 Lua 34
2.8.2 LuaJava 35
2.8.3 PHP 35
2.9 Vorangegangene Arbeiten 36
2.9.1 RHEUMexpert-I 37
2.9.2 RheumaNet 37
2.9.3 RHEUMexpert-II 38
2.9.4 RHEUMexpert/Web 38
2.10 Schnittstellen zu aktuellen Arbeiten 39
2.10.1 Wissenserwerbssystem für RHEUMexpert/Web 39
2.10.2 Entwicklung und Wartung wissensbasierter Systeme . 39

3 Methoden / Anwendung 41
3.1 Definition der neuen Grammatik 43
3.2 Umwandlung der XML-Regeln nach Lua 45
3.3 Generelle Funktionsbeschreibung 45
3.3.1 Operatorfunktionen der ersten Schritte 47
3.3.2 Ergänzende Funktionen 49
3.4 Schritt 0: Definitionen 51
3.5 Schritt 1: Interpretation der Eingaben 52
3.6 Schritt 2: Symptomfindung 54
3.7 Schritt 3: Sicherheitsfaktoren 57
3.8 Schritt 4: Korrektur der Sicherheitsfaktoren 60
3.9 Schritt 5: Auswahl der Verdachtsdiagnosen 61
3.9.1 Konstanter Schwellwert 62
3.9.2 Mittelwert-Algorithmus 63

3.9.3 RHEUMexpert-II QuasiGradientenAlgorithmus 65
3.9.4 Gradientenverfahren 67
3.9.5 Otsu-Algorithmus 69
3.9.6 Vereinfachter Otsu-Algorithmus 69
3.9.7 Zusammenfassung 71
3.10 Einbinden der Lua-Regeln in Java 72
3.11 Alte versus neue Wissensverarbeitung 76
3.12 Conclusio 77

4 Resultate **78**

5 Ausblick **80**
5.1 Begutachtung der Korrekturfaktoren 80
5.2 Veränderbarkeit des Anzeigealgorithmus 80
5.3 Lesbarere Regeln 80
5.4 Fuzzy-Logik 81
5.5 Automatisches Lernen 81

A XML-Lua-Transformationscode **82**

B Lua-Inferenzmaschine **88**

C Neue Lua-Regeln **100**

Literaturverzeichnis **134**

Abbildungsverzeichnis

2.1 Darstellung der Teilgebiete der Künstlichen Intelligenz 10
2.2 Gegenüberstellung: Expertensysteme – Standardprogramme . 11
2.3 Standardaufbau eines Expertensystems 15
3.1 Vereinfachte Darstellung der Inferenz 42
3.2 Komplexere Darstellung der Inferenz 43
3.3 Arbeitsweise des Mittelwert-Algorithmus 63
3.4 Verschiebung des Mittelwerts 64
3.5 Arbeitsweise des Gradienten-Algorithmus 67
3.6 Gradienten-Algorithmus bei gleichen Gradienten 67
3.7 Extremfälle des Gradienten-Algorithmus 68
3.8 Vorgehensweise des vereinfachten Otsu-Algorithmus 70
3.9 Aufbau und Abarbeitung des alten RHEUMexpert/Web . . . 76
3.10 Aufbau und Abarbeitung des neuen RHEUMexpert/Web . . 77

Programmlistings

2.1 Darstellung der ersten Regel zur Veranschaulichung der EBNF 33
3.1 Der Aufbau einer Lua-Regel-Funktion 46
3.2 Initialisierung der Eingabevariablen 51
3.3 Regel 625 des ersten Schrittes in der alten XML-Form 52
3.4 Regel 625 des ersten Schrittes in der neuen Lua-Form 53
3.5 Regel 36 des zweiten Schrittes in der alten XML-Form 54
3.6 Regel 36 des zweiten Schrittes in der neuen Lua-Form 56
3.7 Darstellung der alten XML-Sicherheitsfaktor-Matrix 59
3.8 Darstellung der neuen Lua-Sicherheitsfaktor-Abarbeitung . . 60
3.9 XML-Ergebnis der Inferenz 73
3.10 Alter Java-Server-Quellcode 74
3.11 Neuer Java-Server-Quellcode 75
A.1 PHP-Quellcode zur Umwandlung der XML-Regeln 82
B.1 Lua-Funktionen der Inferenz 88
C.1 Lua-Quelltext der Regeln des zweiten Schrittes 100

1 Einleitung

RHEUMexpert/Web ist ein medizinisches Expertensystem zur Erstellung von Verdachtsdiagnosen in der Rheumatologie. Die früheren Programmversionen von RHEUMexpert hatten keine Möglichkeit die Wissensbasis – aufgrund welcher das Programm Verdachtsdiagnosen ausgibt – anzupassen, da diese direkt im Quellcode enthalten war. Im Zuge der vorliegenden Arbeit wird eine neue formale Grammatik für die Wissensverarbeitung von RHEUMexpert eingeführt und eine Auftrennung von Programmcode und Wissensbasis vollzogen.

1.1 Themengebiet

RHEUMexpert ist ein Konsultationssystem, welches dem Gebiet der Anwendung von Methoden der Künstlichen Intelligenz in der Medizin angehört. Es stellt Verdachtsdiagnosen im Bereich der Rheumatologie um Allgemeinmediziner zu unterstützen, wodurch das breite Themengebiet der Medizin angeschnitten wird. Zum Bau der Wissensmaschine wird auf die Skriptingsprache Lua zurückgegriffen, welche die Abarbeitung in einer virtuellen Maschine ansiedelt. Hierdurch wird auch das Gebiet der interpretierten Programmiersprachen tangiert. Der Bereich der mehrwertigen Logiken ist durch die erstellten Regeln genauso wichtig, wie die informatischen Kostenfunktionen und Testverfahren für die entwickelten Algorithmen. Somit reicht das System RHEUMexpert in vielerlei Themengebiete der medizinischen Informatik.

1.2 Gliederung

- Das erste Kapitel umfasst eine Einführung in die vorliegende Arbeit. Es enthält:
  - die Aufzählung der verwandten Themengebiete,
  - eine Gliederung dieser Arbeit,
  - die Motivation des Autors für diese Arbeit,

  - eine Einführung in die Rheumatologie und
  - das Ziel von RHEUMexpert und dieser Arbeit.

- Das zweite Kapitel beinhaltet eine kurze theoretische Einführung in die wichtigsten Themengebiete, die zum Verständnis des Dokumentes erforderlich sind. Diese sind:
  - Eine Einführung in die Künstliche Intelligenz mit einer Überleitung zum Themengebiet der (medizinischen) Expertensysteme,
  - das Prinzip der Sicherheitsfaktoren und deren Anwendung in der Inferenzmaschine,
  - Grundzüge kompilierter und interpretierter Programmiersprachen und ein Vergleich dieser beiden Sprachfamilien,
  - Eigenschaften von Logiken,
  - informatische Kostenfunktionen,
  - eine Einführung in Grammatiken,
  - die verwendeten Softwarepakete und
  - einen Überblick von Arbeiten zum Thema RHEUMexpert.
- Das dritte Kapitel umfasst die Beschreibung des praktischen Teils der vorliegenden Arbeit, worin die durchgeführten Definitionen und Änderungen an der neuen Wissensrepräsentation sichtbar werden. Es gliedert sich wie folgt:
  - Anfangs wird die formale Definition der neuen Sprache gegeben,
  - danach wird die Umwandlung der alten Regeln in die neue interpretierte Lua-Form dargestellt,
  - eine Beschreibung der erstellten Arithmetik- und Logikoperatoren,
  - die eigentliche Inferenz und
  - die Einbindung der neuen Inferenz in den bestehenden RHEUMexpert/Web-Java-Server komplettieren dieses Kapitel.

- Im vierten Kapitel werden die *Resultate* der vorliegenden Arbeit diskutiert.
- Abschließend wird im fünften Kapitel ein *Ausblick* für Weiterentwicklungen angeführt, um Hinweise für weitere Arbeiten an einer zentralen Stelle anzubieten.

1.3 Motivation

Durch mein Interesse am Forschungsfeld der Künstlichen Intelligenz, und dessen Teilgebiet der Experten- und Konsultationssysteme, habe ich meine Diplomarbeit diesem Themenbereich gewidmet. Da ich schon zwei Praktika im Zusammenhang mit dem medizinischen Expertensystem RHEUMexpert durchgeführt hatte, fiel meine Wahl auf das Institut für Experten- und Wissensbasierte Systeme der Besonderen Einrichtung für Medizinische Statistik und Informatik.

Ein weiteres meiner Interessensgebiete, nämlich die Abarbeitung von Wissen, grenzte das Thema der Magisterarbeit auf eine neuartige Wissensdarstellung für RHEUMexpert ein. Meine Betreuer, Dr. Rudolf Seising und Dipl.-Ing. Reinhard Pitsch, wünschten sich, die komplette Wissensverarbeitung von RHEUMexpert in einer eigenen Grammatik ablaufen zu lassen, da die bestehende Verarbeitung von RHEUMexpert nur ungenügend gut veränderlich und darstellbar war. Weiters war die Adaptier- und Veränderbarkeit von Algorithmen gewünscht, die die Auswahl der anzuzeigenden Verdachtsdiagnosen steuerten.

Mein Kollege Udo Kronberger, der auch auf der Suche nach einem geeigneten Thema für seine Magisterarbeit war, interessierte sich vor allem für die graphische Darstellung einer solchen neuen Wissensrepräsentation, welche auch Thema seiner Arbeit ist. Das Thema der vorliegenden Arbeit richtet sich daher auf die theoretische Aufarbeitung der bestehenden Wissensbasis, einer geeigneten Grammatik und derer Abarbeitung in einer neuen Wissensmaschine.

Erste Ideen, die komplette Verarbeitung in einer vollständig selbst geschriebenen Maschine – so wie RHEUMexpert-II/Web dies bisher tat – ablaufen

zu lassen, wurden aus Performance-Gründen wieder verworfen. Statt dessen wurde der Ansatz gewählt, die Abarbeitung in einer simplifizierten Form der Skriptingsprache Lua – welche mir aus der Spieleprogrammierung bekannt war – ablaufen zu lassen. Nun ging es daran eine möglichst einfache Syntax zu entwickeln, die zwar das Regel-Wissen aus RHEUMexpert abbilden konnte, dennoch aber in dem graphischen Wissenseditor von Kronberger darstell- und editierbar war.

Die vorliegende Arbeit stellt die Ergebnisse dieser Entwicklungsarbeiten dar.

1.4 Klinische Grundlagen

Mit Rheuma (griechisch für „Fließen“, „Strömen“ oder „Ziehen“) werden Beschwerden am Bewegungsapparat, mit fliessenden, reissenden und ziehenden Schmerzen bezeichnet. Heute verstehen wir darunter mehr als 400 Krankheiten, die nicht durch eine Verletzung oder durch tumoröse Veränderungen hervorgerufen worden sind.

Rheuma ist somit weder eine Diagnose im engeren Sinn noch eine einheitliche Krankheit. Vielmehr fallen unter diesen Oberbegriff einzelne Erkrankungen, die sich teilweise ähneln, die aber zum Teil auch völlig unterschiedlich sowohl in ihrer Ursache, der Art ihrer Symptome, aber auch in ihrem Verlauf, ihrer Behandlung und ihren Folgen sind.

Es werden vier große Hauptgruppen unterschieden:

- Degenerative Gelenk- und Wirbelsäulenerkrankungen
  z. B. Arthrose – degenerative Gelenkerkrankung
- Entzündlich-rheumatische Erkrankungen
  z. B. Arthritis – Gelenksentzündung
- Weichteilrheumatismus
  z. B. Fibromyalgie – chronische Schmerzerkrankung in Muskeln oder Sehnen
- Stoffwechselerkrankungen mit rheumatischen Beschwerden
  z. B. Gicht – Ablagerungen in den Gelenken

Anhand dieser Hauptgruppen ist ersichtlich, dass sich rheumatische Erkrankungen nicht allein auf den Bewegungsapparat beschränken, sondern vielmehr Bindegewebsstrukturen im ganzen Körper an rheumatischen Erkrankungen beteiligt sein können.

In leichten Fällen reichen schon einfache Behandlungsmethoden wie Wärme, Kälte, oder Massagen aus. Bei schwereren Fällen sollte in jedem Fall ein Hausarzt oder ein Rheumatologe hinzugezogen werden. Die Behandlung erfolgt dann durch Medikamente (entzündungshemmend, schmerzstillend) und Physiotherapie. Bei starken Beschwerden in Folge von bereits fortgeschrittenem Gelenkverschleiß kommen auch operative Therapien bis hin zum Gelenkersatz in Betracht.

Rheuma ist nicht als eine Krankheit des Alters zu interpretieren. Rheumatische Erkrankungen gibt es auch bei Kindern. Sehr schwere rheumatische Erkrankungen mit lebensgefährlichen Organbeteiligungen können sogar bereits Kleinkinder befallen.

Viele rheumatische Erkrankungen äußern sich durch den typischen rheumatischen Schmerz, der vor allem in Ruhe auftritt oder stärker wird, der in der Nacht den Schlaf stört, sich aber durch Bewegung bessert. Charakteristisch ist auch eine ausgeprägte und länger anhaltende Morgensteifigkeit. Sind die Gelenke betroffen, so können Schwellungen bis hin zur Ausbildung eines Gelenkergusses typische Zeichen für eine rheumatische Entzündung sein.

1.5 Ziel von RHEUMexpert/Web

Im Zuge einer betreuten wissenschaftlichen Arbeit ist die Planung und Entwicklung eines Expertensystems zur Unterstützung von praktizierenden Allgemeinmedizinern bei der Dokumentation und bei ihren differentialdiagnostischen Überlegungen im Bereich der Rheumatologie durchzuführen. Das System soll in vorhandene elektronische Datenverarbeitungssysteme eingliederbar sein. Über die grafische Benutzerschnittstelle soll die Konsultation pro Abfrage außerdem nicht länger als drei bis vier Minuten dauern (gerechnet vom Beginn der Eingabe der vom System erfassbaren Beobachtungen bis zum Erhalt des berechneten Ergebnisses). Der Facharzt für rheumati-

sche Erkrankungen soll und kann durch dieses Programm keinesfalls ersetzt werden. Vielmehr soll bei gegebener Indikation eine Überweisung des Patienten von einem Allgemeinmediziner zu einem Facharzt für rheumatische Erkrankungen erfolgen. Der Einsatz des Expertensystems soll dabei zu einer verbesserten Erkennung von beim Patienten noch nicht diagnostizierten rheumatischen Krankheiten führen.

Das Expertensystem stellt eine Reimplementierung des bereits bestehenden wissensbasierten Systems RHEUMexpert-II dar, welches auch eine Erklärungskomponente aufweist, und eine Erweiterung desselben um die Komponenten Wissenserwerb und Evaluierung. Das System ist plattformunabhängig und als internetfähige Browser-Applikation zu implementieren; Eingabe- und Ergebnisdaten können entweder gedruckt oder gespeichert werden. Insgesamt werden von dem zu erstellenden System die Funktionen Konsultation, Evaluierung und Wissenserwerb angeboten: Die Konsultation steht jedem Benutzer frei und ermöglicht die Eingabe und Auswertung von rheumatische Erkrankungen betreffenden Daten, wobei auf Wunsch das Zustandekommen der Ergebnisse aufgezeigt wird. Die grafische Benutzerschnittstelle orientiert sich an der bereits bestehenden von RHEUMexpert-II.

Die Evaluierung ist einer ausgewählten Benutzergruppe vorbehalten und ermöglicht das Testen der Wissensbasis gegen Goldstandard-Datensätze, wobei als Ergebnis eine Aussage über die Wertigkeit der Wissensbasis getroffen wird. Die Wissensbasis kann modifiziert und für weitere Tests herangezogen werden. Der Wissenserwerb ist ebenfalls einer ausgewählten Benutzergruppe vorbehalten und ermöglicht die Darstellung, Modifikation und persistente Speicherung des gesamten in der Wissensbasis vorhandenen Wissens.

1.6 Ziel der vorliegenden Arbeit

Im Zuge dieser wissenschaftlichen Arbeit ist die Planung und Entwicklung der Wissensverarbeitungskomponente von RHEUMexpert/Web durchzuführen. Hierbei soll ein in einer eigenen Grammatik abarbeitbares System erzeugt werden, welches auf Regeln und Sicherheitsfaktoren basiert.

Die Verarbeitung eines Patientenfalles soll in einer Serverumgebung stattfinden, welche viele Clientanfragen gleichzeitig beantworten kann. Das Interface zwischen dem Server und der ausgelagerten Wissensverarbeitung soll keine Geschwindigkeitseinbussen mit sich bringen. Im Zuge dieser Arbeit sollen auch die in RHEUMexpert/Web verwendeten Algorithmen zum Erlangen der Verdachtsdiagnosen überprüft werden. RHEUMexpert soll gegebenenfalls um neue Algorithmen erweitert werden. Hierbei soll später der untersuchende Wissenschafter (nicht der praktische Arzt!) die Möglichkeit zur Auswahl des geeignetsten Algorithmus haben.

Es sollen erstmals alle Schritte der Wissensverarbeitung (Interpretation der Eingaben, Zuordnung der Interpretationen zu Symptomen, Sicherheitsfaktorberechnung, Korrekturfaktorberechnung und Berechnung ob Verdachtsdiagnosen angezeigt werden sollen) ausserhalb des kompilierten Quellcodes geschrieben sein, um dadurch auch zur Laufzeit des Wissensservers, ohne Abbruch desselben, verändert werden zu können.

Eine formale Definition der neuen Sprache erlaubt interessierten Lesern den tieferen Einblick in die Wissensverarbeitung von Expertensystemen im Allgemeinen und in weitere Publikationen über RHEUMexpert im Speziellen.

2 Material / Verwandte Themengebiete

Das Ziel dieser Magisterarbeit ist es sämtliche Schritte der Diagnosefindung von RHEUMexpert/Web auszulagern und in die Skriptingsprache Lua zu überführen. Es werden weiters zusätzliche Algorithmen konzipiert – und umgesetzt – welche die Auswahl von Verdachtsdiagnosen flexibler gestalten sollen als dies bisher der Fall war. Dadurch verbindet diese Arbeit unterschiedlichste Fachbereiche der Informatik. Um die interessierten Leser im Text nicht mit informatischen Fachbegriffen zu verwirren, werden im folgenden Abschnitt verwandte Themengebiete aufgelistet und die theoretischen Grundlagen zusammengefasst.

Am Beginn steht die Beschreibung einiger Aspekte der Künstlichen Intelligenz, vor allem der Expertensysteme, die für den Leser als Einstieg in das Gebiet der medizinischen Expertensysteme dienen sollen.

Nach dem Abschnitt über Sicherheitsfaktoren – welche in der Diagnosefindung von RHEUMexpert eine wichtige Rolle spielen – wird auf die Arbeit der neuen Wissensverarbeitung und den Unterschied zwischen kompilierten und zur Laufzeit ausgeführten Sprachen eingegangen. Es folgt eine kurze Einführung in die Logik und die Grammatik der alten Wissensverarbeitung mit einer Erklärung der Sprache, in welcher sie geschrieben ist.

Gegen Ende des Kapitels werden die verwendeten Softwarepakete behandelt. Der letzte Punkt befasst sich mit aktuellen und vorausgegangenen Publikationen über RHEUMexpert.

2.1 Künstliche Intelligenz

Der Begriff Künstliche Intelligenz (KI) (engl. Artificial Intelligence bzw. AI) stellt ein breites Anwendungsgebiet der Informatik dar. Gegenstand der KI ist die Erforschung intelligenten Problemlösungsverhaltens und darauf aufbauend die Entwicklung intelligenter Computersysteme. Die KI versucht dabei Eigenschaften und Fähigkeiten mit denen man menschliche Verhaltensweisen verknüpft, z. B. Problemlösungen, Lernen, Erklären und dergleichen auf dem Computer nachzuempfinden oder diese zu simulieren.

Neben den Forschungsergebnissen der Kerninformatik selbst sind in die KI Ergebnisse der Psychologie und Neurologie, Mathematik und Logik, Kommunikationswissenschaft, Philosophie und Linguistik eingeflossen. Der Einfluss der Neurologie hat sich in der Entstehung des Bereichs Neuroinformatik gezeigt, der der bioorientierten Informatik zugeordnet ist. Zusätzlich ist auch der komplette Zweig der Kognitionswissenschaft zu nennen, welcher sich wesentlich auf die Ergebnisse der KI in Zusammenarbeit mit der kognitiven Psychologie stützt.

In [13] zieht der Philosoph Searle aus einem Gedankenexperiment – dem so genannten „Chinesischen Zimmer“, welches Alan Turings Versuch „denkende Maschinen“ einzugrenzen zu widerlegen versucht – die Konsequenz, dass zwischen einer schwachen und einer starken KI unterschieden werden muss. Die schwache KI versucht menschliches Verhalten zu simulieren und Probleme zu lösen, die von Menschen nur mittels Intelligenz zu bewältigen sind. Vertreter der starken KI möchten hingegen denkende Maschinen bauen, was nach Searle unmöglich ist.

2.1.1 Intelligente Systeme

Bei intelligenten Systemen geht es darum konkrete Anwendungsprobleme zu meistern. Sie sind Teil des Forschungsgebietes der schwachen KI. Insbesondere sind dabei solche Anwendungen von Interesse, zu deren Lösung nach allgemeinem Verständnis eine Form von „Intelligenz“ notwendig zu sein scheint. Letztlich geht es aber um die Simulation intelligenten Verhaltens mit Mitteln der Mathematik, Statistik und der Informatik.

Hierbei geht es nicht um die Schaffung von Bewusstsein oder um ein tieferes Verständnis von Intelligenz. Während die starke KI in genau diesen Bereichen noch immer nur geringe Erfolge aufweist und an ihrer philosophischen Fragestellung nach einem Bewusstsein in Maschinen bis heute scheitert [13], sind auf der Seite der schwachen KI in der Vergangenheit schon bedeutende Fortschritte erzielt worden.

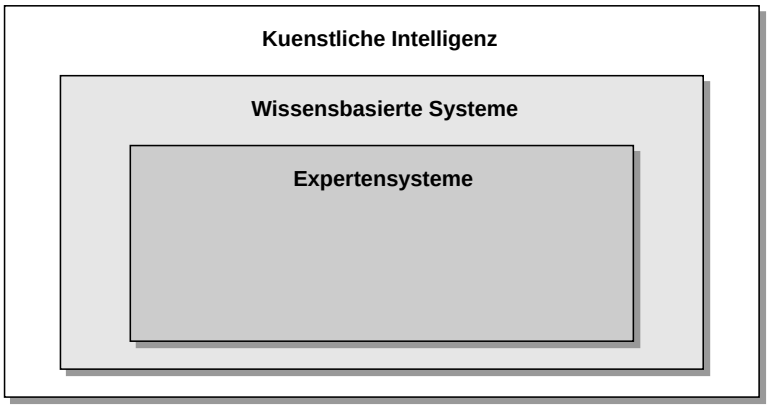

Abbildung 2.1: Darstellung der Teilgebiete der Künstlichen Intelligenz

2.1.2 Wissensbasierte Systeme

Nach Waterman [17] sind Expertensysteme eine Teilmenge der wissensbasierten Systeme, welche wiederum ein Teilgebiet der KI sind, was in Abbildung 2.1 ersichtlich ist. Der Unterschied zwischen wissensbasierten Systemen und Expertensystemen liegt im dezidiert gespeicherten Expertenwissen, und nicht nur, wie bei wissensbasierten Systemen, in der Abtrennung des Wissens (Wissensbasis) vom restlichen und unabhängigen Problemlösungsverhalten des Programmes. Expertensysteme sollten eine vollständige Expertise eines Experten enthalten, d.h. sowohl sein Fach- als auch sein Erfahrungswissen. Andere Systeme aus den verschiedenen Gebieten der KI verarbeiten in der einen oder anderen Weise Wissen, nicht aber Expertenwissen, wie dies gerade in der Medizin erforderlich ist.

Wie man in Abbildung 2.2 erkennt, unterscheiden sich Expertensysteme in deren Aufbau stark von anderen Programmsystemen. Besonderes Augenmerk wollen wir auf die vorhin schon beschriebende Unterteilung des Expertensystems in Expertenwissen und Problemlösungsstrategie (Inferenzmaschine) legen.

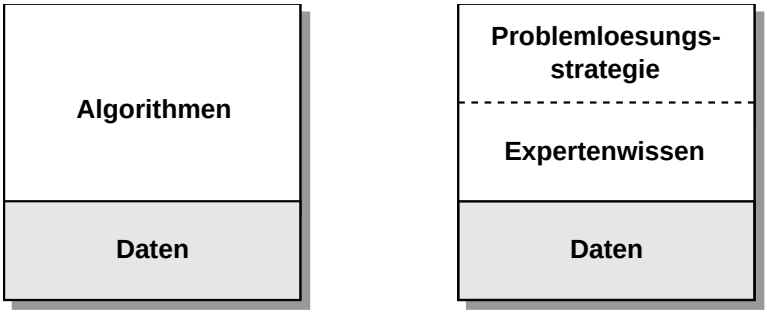

Abbildung 2.2: Gegenüberstellung: Expertensysteme – Standardprogramme

2.1.3 Expertensysteme

Expertensysteme sind jene Programme, die das Wissen und die Erfahrung menschlicher Experten zur Lösung von Aufgaben eines bestimmten Problembereiches nachahmen können. Ein Experte in einem abgegrenzten medizinischen Gebiet hat detaillierte Kenntnisse (auf diesem Gebiet) und hervorragende Eigenschaften mit diesen Kenntnissen effizient umzugehen. Er weiss sein medizinisches Wissen bezüglich

- des Umfangs (was zu tun ist)
- der Art und Weise (wie es zu tun ist)
- der Zeit (wann etwas zu tun ist)
- des Ortes (wo es zu tun ist)
- der Notwendigkeit (warum es zu tun ist)

einzusetzen [12].

Im Fall von Expertensystemen versucht man, die Fähigkeit zur Problemlösung, in diesen kleinen Arbeitsbereichen, derer menschlicher Experten nachzuempfinden. Die Hauptaufgabe von Expertensystemen in der medizinischen Informatik liegt in der Abarbeitung von Fragestellungen, die von Experten routinemässig beantwortet werden können, für die allerdings sowohl Algorithmen, als auch statistische Daten fehlen.

Expertensysteme sollen aber auch negative Eigenschaften von Experten eleminieren. Diese wären unter anderem deren geringe Anzahl (in manchen Fällen gibt es nicht mehr als fünf Experten weltweit), deren begrenzte Zeit und Geduld, deren Tagesverfassung und vor allem deren Vergänglichkeit.

Ein anderer Vorteil von Expertensystemen ist, dass Studenten ihr Wissen weltweit an diesen Systemen testen können und auf diesem Weg – mit geeigneten graphischen Mitteln – ein gesteigerter Lernerfolg zu erwarten ist.

Expertensysteme sind das bisher erfolgreichste Anwendungsgebiet der KI.

2.1.4 Erfolgreiche medizinische Expertensysteme

Hier soll nun eine kurze Liste von (in der einen oder anderen Art) erfolgreichen medizinischen Expertensystemen dargestellt werden. Deren Erfolg schlägt sich zum Teil nicht in der praktischen Anwendung, sondern vielmehr in den richtungsweisenden Ideen nieder, die diesen Expertensystemen zugrunde liegen.

MYCIN
MYCIN [3, 15] ist eines der ersten und wichtigsten medizinischen Expertensysteme. Es wurde entwickelt um Erreger von Infektionskrankheiten zu bestimmen und entsprechende therapeutische Massnahmen vorzuschlagen. Aus heutiger Sicht wurde mit MYCIN der Grundstein für viele weitere Expertensysteme gelegt; schon allein wegen der wissenschaftlichen Erkenntnisse, die aus der Entwicklung im Bereich der Sicherheitsfaktoren, der Inferenzmethodik, dem (halbautomatischen) Wissenserwerb und der Expertensystem-Shells gezogen wurden. Das medizinische Wissen wurde später aus MYCIN entfernt, wodurch die Inferenzmaschine bzw. die Expertensystem-Shell EMYCIN (Essential MYCIN scherzhaft auch als Empty MYCIN bezeichnet) zurück blieb und in anderen (teils auch nicht-medizinischen) Expertensystemen zum Einsatz kommt [10].

PUFF
PUFF ist ein Expertensystem, welches Lungenfunktionsdaten interpretiert und diagnostiziert. Es wurde für das Pacific Medical Center in San Francisco erstellt und ist dort (in einigen Adaptionen) noch im Einsatz. PUFF ist ein Beispiel für Expertensysteme, die – wie im vorigen Kapitel beschrieben – auf EMYCIN aufbauen. Die Entwicklungszeit von PUFF konnte im Gegensatz zu MYCIN, je nach einzelnen Quellen in der Literatur, um das vier- bis zehnfache verkürzt werden [1].

INTERNIST / CADUCEUS

Die derzeit größte Wissensbasis eines medizinischen Expertensystems ist die von INTERNIST. Sie besteht aus 24.000 Inferenzregeln, die durch weitere 3.000 Einheiten referentiellen Wissens und 400 Einheiten begrifflichen Wissens ergänzt werden [33]. Es ist derzeit das ultimative diagnostizierende Expertensystem und soll das komplette Fachgebiet der Inneren Medizin abbilden. Man schätzt, dass in den letzten 25 Jahren ungefähr 80% des medizinischen Wissens der Inneren Medizin in INTERNIST bzw. CADUCEUS eingefügt wurden. [32].

Im Wissenssystem von INTERNIST ist die bedeutendste Struktur das Krankheitsprofil, in welchem jeder Krankheit Symptome und Befunde zugeordnet werden, die bei dieser Krankheit auftreten können. Ausgehend von der Menge der Krankheitsprofile wird für jedes der dort vorkommenden Symptome ein inverses Krankheitsprofil gebildet. Hierbei werden jedem Symptom alle Krankheiten, in denen es aufgrund der Krankheitsprofile vorkommt, zugeordnet [12, Seite 116].

CASNET

Die erste Nutzung eines anderen Ansatzes – nämlich den des kausalen-assoziativen Netzes – stellt CASNET (engl. Abk. Causal-Associational Network) dar [34]. Das System gibt Hinweise auf diagnostische, therapeutische und prognostische Hinweise zum Stadium des Glaukoms – der auch als Grüner Star bezeichneten häufigsten Erkrankung des Sehnervs – eines Patienten. Es stellt Hypothesen über pathophysiologische Zustände im Auge auf und versucht diese zu beweisen oder zu widerlegen indem es die Ergebnisse der den Zuständen assoziierten Tests anfordert.

Diese Symptome und Befunde, mögliche pathophysiologische Zustände und vielerlei Krankheitskategorien werden in der Wissensbasis unterschieden. Für diese drei Objekttypen wird ein Netzwerk bestimmt. Die Beziehung zwischen den Observationen und pathophysiologischen Zuständen sind assoziativ, die innerhalb dieser Zustände sind kausal und jene zwischen den Zuständen und Krankheitskategorien sind klassifizierend. CASNET ist hierdurch das erste Expertensystem, welches versucht verschiedene Arten des Wissens (ähnlich denen eines Mediziners) nachzuempfinden [43].

CADIAG

CADIAG-I–IV (engl. Abk. Computer Aided Diagnosis) sind diagnostische Konsultationssysteme für die Medizin [21]. CADIAG-I verwendet Relationen, alle weiteren Systeme sind regelbasiert und verwenden Fuzzy-Logik [20, 22]. Sie wurden am Institut für Medizinische Computerwissenschaften (IMC) der medizinischen Fakultät der Universität Wien entwickelt und bilden die Grundlage für eine Reihe weiterentwickelter Expertensysteme [31]. Nähere Informationen zur Fuzzy-Logik finden sich in Kapitel 2.5.3.

2.1.5 Arbeitsweise

Ein Expertensystem verfügt über eine Architektur, die aus einer Wissensbasis, einer Wissensverarbeitung, einer Erklärungs-, einer Wissenserwerbskomponente und aus Schnittstellen zwischen diesen besteht, wie in Abbildung 2.3 erkannt werden kann. Pitsch führt hierfür in [39, Seite 29ff.] die Begriffe Konsultation, Evaluation und Wissenserwerb ein, die auch in der vorliegenden Arbeit verwendet werden. Die jeweiligen Teilsysteme – die zum Zeitpunkt der Drucklegung von Pitschs Arbeit nur ansatzweise existierten – werden im Kapitel 3.11 mit deren aktuellem Stand angeführt.

Wie schon im vorigen Unterkapitel zu lesen war, befindet sich das Wissen in der Wissensbasis zumeist in Form von Regeln. Die Wissensverarbeitungskomponente stellt Mechanismen bereit, mit deren Hilfe neues Wissen abgeleitet werden kann. Die Wissenserwerbskomponente dient der adäquaten Darstellung des gespeicherten Wissens und stellt Modifikationsmöglichkeiten der Wissensbasis zur Verfügung. Mit Hilfe einer solchen Komponente kann das Wissen des Systems erweitert, verändert oder auch reduziert werden. Die Erklärungskomponente dient der schlüssigen und verständlichen Aufbereitung des Lösungsweges, den das Expertensystem gewählt hat, um zu einem errechneten Ergebnis zu gelangen. Diese beiden Komponenten – und der Ansatz deren Umsetzung – werden näher in Kronbergers Magisterarbeit beschrieben.

In Kapitel 3.11 Abbildung 3.9 und 3.10 ist eine Gegenüberstellung der graphischen Darstellungen des alten- und des neuen RHEUMexpert/Web zu sehen.

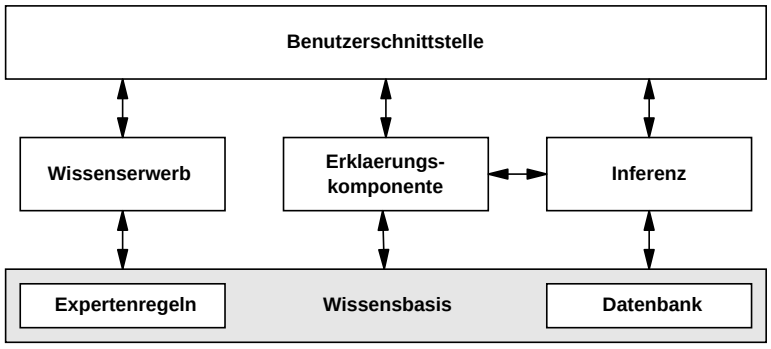

Abbildung 2.3: Standardaufbau eines Expertensystems

2.2 Sicherheitsfaktoren

Die erste Verwendung fanden Sicherheitsfaktoren (engl. Certainty Factors, in mancher Literatur auch als Gewissheitsfaktoren bezeichnet) im regelbasierten Expertensystem MYCIN [3, 12, Seite 87], für welches sie auch entwickelt wurden. Ziel war es, Expertenmeinungen in Form von Regeln mit einer gewissen Sicherheit nachzuahmen.

Der Begriff Certainty Factors wurde 1975 von Shortliffe und Buchanan eingeführt. Die Autoren wollten damit eine Alternative zur Bayes'schen Wahrscheinlichkeitstheorie anbieten, da diese für den Einsatz in Expertensystemen nur bedingt geeignet war [3, Seite 236f.]. So steht etwa die nahezu unüberschaubare Menge an benötigten Daten über den Patienten nur selten zur Gänze zur Verfügung.

Edwards erkannte 1972, dass es sinnvoller ist Expertenwissen zu akquirieren anstatt statistische Auswertungen durchzuführen [16],[15, Seite 236ff.]. Befragt man viele Experten zu gewissen Krankheiten, entsprechender Symptome und deren Verknüpfungen, so bekommt man ein weites Spektrum an medizinischem Wissen – Wissen welches Ärzte/Experten in ihrem täglichen Umfeld verwenden. Aus diesem Wissen werden dann Regeln abgeleitet, welche als zentrale Elemente Certainty Factors verwenden. [12, Seite 184ff.],[3, Seite 210ff.]

2.2.1 Definition

Ein Sicherheitsfaktor (CF) ist ein numerischer Wert im Intervall $[-1, +1]$. Er repräsentiert den Grad des Vertrauens in eine Hypothese. Ein positiver CF bestätigt die Hypothese H zu einem gewissen Grad, und ein negativer CF kontraindiziert die Hypothese H. Je größer der CF, desto größer ist der Grad des Vertrauens in die Hypothese. [15, Seite 99ff.] Wenn der CF gleich $+1$ ist, so ist die Hypothese bewiesen und trifft zu. Vice versa wird die Hypothese widerlegt wenn der Wert -1 beträgt. Sollte der CF den Wert 0 annehmen, kann keinerlei Aussage über die Hypothese getätigt werden.

2.2.2 Measures of (Dis-)Belief

Der Grad des Vertrauens (MB) (engl. „measure of belief“ und der Grad des Misstrauens (MD) (engl. „measure of disbelief“) tragen gemeinsam zum Sicherheitsfaktor bei.

„The measure of increased belief in the hypothesis h, based on the evidence e, is X. The measure of increased disbelief in the hypothesis h, based on the evidence e, is Y“ [3, Seite 169]

$$CF\,[h,e] = MB\,[h,e] - MD\,[h,e] \tag{1}$$

MB und MD können Werte im Intervall $[0, +1]$ annehmen. Kombiniert man nun beide Werte entsprechend der Gleichung 1 zu dem zugehörigen Certainty Factor, so sind für diesen alle Werte im Intervall $[-1, +1]$ möglich. MB und MD waren nur in der ursprünglichen Implementation der Certainty Factors vorhanden, wurden jedoch in neueren Systemen, z. B. EMYCIN durch eine einzelne Variable (CF) ersetzt [29, Seite 328ff.]. Über die Erfassung der Werte für MB und MD findet man in der Literatur unterschiedlichste Angaben.

2.3 Inferenz

In der Informatik wird die Schlussfolgerung auch gelegentlich mit dem sonst im Deutschen unüblichen Fremdwort „Inferenz“ bezeichnet, vermutlich als Übersetzung des englischen Wortes „inference“ (Schluss, Folgerung). Meist

wird das Wort „Inferenz“ in der Informatik spezieller für solche Schlussfolgerungen verwendet, die automatisiert, d. h. computergestützt, gezogen wurden. Im Bereich der Expertensysteme wird die „Inferenz“ als jener Vorgang beschrieben, der vom Erhalt der (Patienten-)Daten bis zur Rückgabe des Ergebnisses (der Verdachtsdiagnosen) reicht und somit der Abarbeitung der Wissensmaschine entspricht.

Um von einer Inferenzmaschine sprechen zu können benötigt eine solche zumindest folgende Bestandteile:

- eine Wissensbasis,
- Operatoren, welche neues Wissen ableiten können und
- Kontrollstrategien, welche die Anwendung und Reihenfolge der Operatoren steuern aber auch entscheiden welche Verdachtsdiagnosen angezeigt werden.

In der Literatur zu regelbasierten Systemen sind unterschiedliche Sichtweisen beschrieben, wie die Inferenzmethodik regelbasierter Systeme aufgefasst werden soll. Hier sei nun der Ansatz der Inferenz von RHEUMexpert vorgestellt, der die Arbeitsweise des Systems darstellt.

Zu Beginn sind alle Bewertungen auf „nicht definiert“ gesetzt. In der Lua-Umgebung, welche näher im Kapitel 2.8.1 und ab dem Kapitel 3 besprochen wird, entspricht dies dem selbstgewählten Wert `-0,00000001` und wird im Kapitel 2.5.2 über dreiwertige Logiken näher erklärt. Anschliessend werden die im Client eingegebenen Daten vom Server an die LuaJava-Maschine übergeben – die entsprechenden Variablen werden also mit den eingegebenen Daten befüllt. Ab diesem Abschnitt im Programmablauf wird die Inferenz zwar in fünf weitere Schritte unterteilt, diese gehen aber eng miteinander einher und werden ab dem Kapitel 3.4 genauer erörtert. Hier sei nur vermerkt, dass die Eingangsvariablen aufgrund von Regeln zuerst klassifiziert, dann zu Symptomen aufsummiert und mit Hilfe von Sicherheitsfaktoren zu Verdachtsdiagnosen umgeschlüsselt werden. Diese werden in den letzten beiden Schritten korrigiert, und die sichersten Verdachtsdiagnosen – aufgrund von definierten Algorithmen – über ein Interface zwischen der Inferenzmaschine und dem Server wieder zurück an den Client gesendet.

2.4 Programmiersprachen

Im folgenden Unterkapitel sollen einige Grundzüge von Programmiersprachen gezeigt werden. Dies ist nötig, um das Verständnis der Technik hinter der Wissensverarbeitung mit Lua zu gewährleisten.

Eine Programmiersprache ist eine formale Sprache, die zur Erstellung von Anweisungen für Computersysteme verwendet wird. Sie richtet sich deshalb in Form und Funktion als Sprache an die Struktur und Bedeutung von Information. Hierdurch können Anweisungen in einer für den Menschen lesbaren und verständlichen Form notiert werden. Programmiersprachen sind notwendig, da die natürlichen Sprachen oder Zahlensysteme für eine genügend präzise Beschreibung von Algorithmen zu vieldeutig, nicht formal genug oder nicht für den Menschen verständlich sind. Die syntaktische Definition einer Programmiersprache wird meist in der formalen Notation der Backus-Naur-Form angegeben, welche im Kapitel 2.7.1 erläutert wird.

2.4.1 Kompilierte Sprachen

Unter einem kompilierten Programm versteht man in der Informatik das Ergebnis der Anwendung eines Compilers auf den Quelltext eines Computerprogramms. Dabei wird das in einer Quellsprache geschriebene Programm in ein semantisch äquivalentes Programm in der Zielsprache übersetzt.

Bei der Kompilierung eines Computerprogrammes wird meist aus dem Quelltext eine Bibliothek oder ein ausführbares Programm erzeugt, welches je nach Plattform als ausführbare Datei (Executable), Binärdatei oder Lademodul bezeichnet wird. Es kann aber auch je nach Compiler beispielsweise eine Ausgabe in einer anderen Programmiersprache erzeugt werden, die dann weiterverarbeitet wird.

Der Compilerbau und die Optimierung des Quelltextes ist eines der ältesten Forschungsgebiete der Informatik und soll in dieser Arbeit nicht näher erörtert werden.

Dem Kompilieren steht die Interpretation gegenüber, bei der ein Quelltext eines Programmes Anweisung für Anweisung zur Laufzeit ausgeführt wird,

wie im nächsten Kapitel erkennbar ist. Weitere Informationen können beispielsweise in [18] erhalten werden.

2.4.2 Interpretierte Sprachen

Ein Interpreter – im Sinne der Informatik – ist eine Software, die den Quellcode eines Programmes im Gegensatz zu Assemblern oder Compilern nicht in eine auf dem System direkt ausführbare Datei umwandelt, sondern den Quellcode einliest, analysiert und ausführt. Die Analyse und Interpretation des Quellcodes erfolgt somit zur Laufzeit des Programms.

Die grössten Nachteile einer interpretierten Sprache sind die im Vergleich zu kompilierten Programmen etwas langsamere Ausführungsgeschwindigkeit und ihre deutlich langsamere Startzeit. Reine Interpreter lesen und analysieren den Quellcode eines Programmes und führen dann die entsprechenden Aktionen durch. Dies ist sehr zeitaufwändig im Vergleich zu Compilersprachen, bei denen das Programm vor seiner Ausführung in Maschinencode übersetzt wird, der dann vom Prozessor direkt ausgeführt wird. Die Vorteile der Interpreter liegen aber darin, dass sie auf jeder Rechnerarchitektur lauffähig sind – wenn der Quellcode des Interpreters (der aus Performance-Gründen häufig in C geschrieben ist) dort übersetzt werden kann – und dass Programme die mit interpretierten Sprachen umgesetzt sind, deutlich verkürzte Entwicklungszyklen (rapid Prototyping) aufweisen. Weiters benötigen interpretierte Programme oftmals weniger Hauptspeicher, da nur der Interpreter selbst und die Rechenschritte des Programmes im Speicher gehalten werden, die zur Anwendungszeit tatsächlich verwendet werden. Compilierte Programme hingegen werden als Ganzes in den Hauptspeicher geladen.

Eine Kompromisslösung ist ein Just-In-Time-Compiler (JIT-Compiler), bei dem das Programm erst zur Laufzeit, jedoch direkt in Maschinencode, übersetzt wird. Danach wird der übersetzte Code direkt vom Prozessor ausgeführt. Durch Zwischenspeicherung des Maschinencode müssen mehrfach durchlaufene Programmteile nur einmal übersetzt werden. Auch ermöglicht der JIT-Compiler eine stärkere Optimierung des Binärcodes eines speziellen Computersystems. Allerdings sind solche Interpreter zur Laufzeit nur auf einer bestimmten Rechnerarchitektur lauffähig, weil sie Maschinencode für

diese Architektur erzeugen. Dies ist aber kein Problem, da der Quelltext, wenn er auf ein anderes Computersystem übertragen wird, automatisch bei der ersten Ausführung neu übersetzt wird.

Eine weitere Zwischenstufe sind Bytecode-Interpreter. Dabei wird der Quelltext zur Laufzeit vor seiner Ausführung in einen einfachen Zwischencode übersetzt, der dann von einem Interpreter – häufig als virtuelle Maschine bezeichnet – ausgeführt wird. Näheres dazu im Kapitel 2.4.3. Bekannte Programmiersprachen, die üblicherweise in Bytecode übersetzt werden, sind Java, C#, Perl und Python.

Bekannte Interpretersprachen sind BASIC, Perl, PHP, Lua und viele andere. Auch Skriptsprachen wie z. B. Javascript oder MS Office Makros können zu den Interpretersprachen gezählt werden. Für manche Sprachen wie etwa Smalltalk oder Lua gibt es aus Performance-Gründen und je nach Anwendungsgebiet Interpreter, Bytecode-Interpreter, JIT-Compiler oder Compiler, die die Programme in andere Sprachen oder ausführbaren Maschinencode übersetzen.

Eine Entwicklung der letzten Jahre ist es, dass der Übergang zwischen Interpretern und Compilern mit neuartigen Ahead-Of-Time-Compilern immer fliessender wird, und der Quelltext so eingesetzt werden kann, wie es die Anwendung erfordert.

2.4.3 Virtuelle Maschinen

Virtuelle Maschinen (VM) stellen in der Softwaretechnik eine virtuelle Laufzeitumgebung für Programme innerhalb eines Host-Systems zur Verfügung.

Sie können entweder vollständig in Software oder mittels einer Kombination aus Software und Hardware implementiert werden. Stellt das Host-System keine Möglichkeit zur Verfügung den Anwendungscode direkt auf einer CPU ausführen zu können, muss ein Interpreter den Anwendungscode indirekt ausführen. Benötigt bei der Virtualisierung von Hardware das Host-System Schnittstellen (z. B. für Geräte und Kommunikation), die das Host-System nicht bereitstellt, müssen diese emuliert werden.

Virtuelle Maschinen mit einer Orientierung zur Softwareausführung spielen

heute eine bedeutende Rolle. Microsoft ist mit der .NET-Architektur dem Beispiel von Sun mit der Java VM, und dem der Entwickler von Lua gefolgt. Hintergrund ist jeweils die Orientierung auf plattformunabhängige Ausführung von Programmen auf verschiedenen Rechnern, die über das Internet verbunden sind.

Die heutigen VM-Systeme besitzen eine große Zahl von Vorläufern, bei denen nutzerorientierte Darstellungen eines Programms (Programmiersprachen) nicht direkt in die maschinenorientierte Darstellung der CPU übersetzt wurden (Maschinencode), sondern in einen einfach strukturierten Zwischencode. Dieser wird dann auf dem Zielsystem durch einen Interpreter ausgeführt. Neben der direkten Ausführung stehen dabei auch verschiedene Optimierungen zur Verfügung, die bereits im vorigen Kapitel erwähnt wurden.

Die Speicherung des Zwischencodes kann unterschiedlich ausfallen, etwa als Bytecode oder als Baumstruktur. Technisch kann dies als Vorkompilierung betrachtet werden, bei der die Analyseschritte eines Compilers zum Verständnis der problemorientierten Programmiersprache durchlaufen werden (Frontend des Compilers), jedoch keine maschinenorientierte Anpassung an eine spezielle CPU erfolgt (Backend des Compilers), sondern die Anpassung an die abstrakten Ausführungseigenschaften der virtuellen Maschine erfolgt.

Bei Interesse für das Forschungsgebiet der virtuellen Maschinen empfiehlt der Autor [19].

2.4.4 Lexen und Parsen

In der Informatik spricht man bei der Zerlegung einer Eingabe in eine Folge von logisch zusammengehörigen Einheiten – den so genannten Token – von einer lexikalischen Analyse oder kurz vom Lexen. Typischerweise geschieht die Zerlegung nach den Regeln einer regulären Grammatik, welche auf eine BNF oder EBNF aufbaut. Der Scanner ist üblicherweise als endlicher Automat realisiert.

Ein Lexer ist ein spezieller Parser und meist vorverarbeitender Teil eines weiteren Parsers. Er erkennt dabei innerhalb der Eingabe Schlüsselwörter, Bezeichner, Operatoren und Konstanten. Erkannte Token werden mit ihrem

jeweiligen Typ zurückgeliefert. Der nachfolgende Parser arbeitet dann auf den Token als atomaren Symbolen (auch Terminalsymbole genannt).

Ein Scanner kann einen separaten, so genannten Screener benutzen, um Leerraum und Kommentare zu entfernen. Häufig wird das jedoch von der zugrunde liegenden Grammatik abgedeckt.

Ein Parser (engl. to parse „analysieren“ bzw. lat. pars „Teil“) ist eine Software, die in der Computertechnik für die Zerlegung und Umwandlung einer beliebigen Eingabe in ein für die Weiterverarbeitung brauchbares Format zuständig ist. Häufig werden Parser eingesetzt, um im Anschluss an den Analysevorgang die Semantik einer Eingabe zu erschließen und daraufhin Aktionen durchzuführen. Zur Analyse des Texts verwenden Parser in der Regel einen separaten lexikalischen Scanner (der wie schon weiter oben genannt als Lexer bezeichnet wird).

Die in der vorliegenden Arbeit noch weiter besprochenen XML-Parser analysieren XML-Dokumente und stellen die darin enthaltenen Informationen – deren Elemente, Attribute usw. – für die weitere Verarbeitung zur Verfügung.

Die Syntaxanalyse (engl. parsing) findet auch außerhalb der Informatik Anwendung, z. B. bei der Untersuchung der Struktur von Sprachen. In der Grammatik würde die Syntaxanalyse eines Satzes dem Zerlegen des Satzes in seine grammatikalischen Bestandteile – der Syntax – entsprechen.

Nähere Informationen zum Thema Lexen und Parsen gibt [6].

2.5 Aussagenlogik

Informatisch gesehen ist die Logik die Lehre der Aussagen und des vernünftigen Schliessens. Die Logik untersucht die Gültigkeit von Ausagen hinsichtlich ihrer Struktur unabhängig vom konkreten Inhalt dieser Aussagen. In diesem Sinne spricht man auch von „formaler“ Logik. Die Logik ist ursprünglich ein Teilgebiet der Philosophie, sie wird heute aber oft auch als Teilgebiet der Mathematik und der Informatik angesehen. [49]

[11] gibt nähere Informationen zum Thema Aussagenlogik.

2.5.1 Zweiwertige Logik

Das Prinzip der Bivalenz ist die Eigenschaft einer Logik, dass jeder Aussage genau einer von zwei Wahrheitswerten zugewiesen wird. Häufig werden diese Wahrheitswerte als `true` und `false` bezeichnet. Hierbei spricht man von wahren oder falschen Aussagen. Logiken, für die das Prinzip der Zweiwertigkeit erfüllt ist, nennt man zweiwertige Logiken.

Aus Aussagen werden hierbei nach festgelegten Vorschriften Formeln gebildet. In einfachen Fällen werden diese mit den logischen Verknüpfungssymbolen Konjunktor ($\wedge$), Disjunktor ($\vee$) oder Negator ($\neg$) verbunden. Zur Interpretation der Formeln werden die Wahrheitswerte `true` bzw. `false` vergeben.

Das Prinzip der Zweiwertigkeit ist zu unterscheiden von dem auch innerhalb mehrerer mehrwertigen Logiken gültigen Satz vom ausgeschlossenen Dritten, der besagt, dass sich $P \vee \neg P$ innerhalb des logischen Systems bzw. seines Kalküls syntaktisch ableiten lässt.

Diese Art der Logik ist in beinahe allen Programmiersprachen realisiert, und meist mit einem Datentyp der auf George Boole verweist gekennzeichnet. Diese Boole'schen Datentypen existieren auch in deren Interpretation in Lua und damit in der erschaffenen Inferenz. Leider ist diese Art der Logik für medizinische Anwendungen ungenügend, weshalb von Boole'schen Variablen in der Verarbeitung der neuen Inferenz Abstand genommen, und diese in der vorliegenden Arbeit durch eine andere Art der Logik ersetzt wurde.

2.5.2 Dreiwertige Logik

Die dreiwertige Logik ist ein Beispiel für mehrwertige Logiken. Dies sind Logiken, die sich von der klassischen Logik dadurch unterscheiden, dass das Prinzip der Zweiwertigkeit aufgegeben wird – dass es also mehr als zwei Wahrheitswerte gibt [9]. Im engeren Sinne versteht man darunter das von Łukasiewicz 1920 vorgestellte System L_3 [30]. In den USA hat zuerst Stephen Kleene dreiwertige Systeme bzw. mehrwertige Logiken eingeführt [7].

Neben den Wahrheitswerten `true` und `false` der klassischen Logik wird ein

dritter Wahrheitswert eingeführt. Bei Łukasiewicz, der von einer erkenntnistheoretischen Fragestellung ausgeht, ist die Bedeutung in etwa: nicht bewiesen, aber auch nicht widerlegt, und kann als *m* – möglich gelesen werden. Interpretationen, die L_3 in der Informatik anwenden, lesen den dritten Wahrheitswert als *u* für unbekannt.

Diese Art der Logik kommt gerade in der medizinischen Informatik oftmals zur Anwendung, da nicht alle Daten eines Patienten aus Komplexitätsgründen zwingend für viele Diagnosen vorhanden sein müssen. Es ist oftmals eine Frage der körperlichen Belastung für den Patienten, aber auch der Kosten für die Allgemeinheit, ob eine Vielzahl von Daten (wie z. B. einer Magnet-Resonanz-Tomographie) erhoben wird. In der heutigen Zeit wird aus Gründen der Wirtschaftlichkeit versucht möglichst wenige Daten zu erheben, aber mit diesen dennoch zu einer korrekten Diagnose zu gelangen. Dies wäre mit Hilfe der zweiwertigen Logik nicht möglich, da ein Wert nicht ausgeschlossen werden kann, nur weil er nicht erhoben wurde.

Im Fall der neuartigen Lua-Inferenz kommt dem Wert `-0,00000001` diese Bedeutung zu. Sämtliche Operatorfunktionen gehen mit unbekannten Werten dieser Art korrekt um, wodurch L_3 in Lua implementiert wurde.

2.5.3 Fuzzy-Logik

Fuzzy-Logik (engl. unscharfe Logik) ist eine Theorie, eine Verallgemeinerung der zweiwertigen Boole'schen Logik, die vor allem für die Darstellung menschlichen – somit unscharfen – Wissens entwickelt wurde.

Fuzzy-Computersysteme verarbeiten gegenüber herkömmlichen Systemen nicht nur Werte wie `true` und `false` (andere Darstellungsformen sind 1 und 0), sondern zusätzlich als Wahrheitswerte zwischen `true` (=1) und `false` (=0) z. B. 0,62 oder 0,37, so dass damit auch unscharfe Angaben wie EIN BISSCHEN, ZIEMLICH oder STARK mathematisch behandelt werden können. Damit arbeiten Fuzzy-Programme näher am menschlichen Denken als herkömmliche Programme.

Die Fuzzy-Set-Theorie, auch als die unscharfe Mengenlehre bekannt, wurde 1965 von Lotfi A. Zadeh an der University of California in Berkeley ent-

wickelt [35]. Sie ist von der – im vorigen Kapitel besprochenen – mehrwertigen Logik zu unterscheiden. Im engeren Sinne kann die Fuzzy-Logik zwar als eine mehrwertige Logik gedeutet werden, und insofern gibt es eine gewisse Nähe zur mehrwertigen Logik, für deren Wahrheitswerte einer logischen Aussage Zahlen aus dem reellen Einheitsintervall [0, 1] (die reellen Zahlen von 0 bis 1) verwendet werden, allerdings fasst Zadeh die Fuzzy-Set-Theorie und Fuzzy-Logik viel weiter, da er das Augenmerk auf die Unschärfe der Mengen legt. [14].

Heute wird die Fuzzy-Logik oder Fuzzy-Control vorwiegend bei der Steuerung von Maschinen und Robotern oder auch handelsüblichen Haushaltsgeräten verwendet, findet aber auch im Bereich der Medizin und der Expertensysteme ihre Anwendung [31, 35, Kapitel 7].

2.6 Landau-Symbole / Kostenfunktionen

Landau-Symbole werden in der Informatik und Mathematik verwendet, um das asymptotische Verhalten von Funktionen und Folgen zu beschreiben. Hierbei werden sie insbesondere in der Komplexitätstheorie verwendet, um verschiedene Algorithmen danach zu vergleichen, wie „schwierig“ oder aufwändig sie im Bezug auf den Speicher (Platzkomplexität) und auf die Zeit (Zeitkomplexität) zu berechnen sind.

Der grosse griechische Buchstabe Omikron $\mathcal{O}$ und klein o sind die am häufigsten verwendeten Landau-Symbole. Darüber hinaus gibt es noch groß Omega (Ω), klein omega (ω) und Theta (θ). Sie vergleichen das Wachstum von zwei Funktionen, meist im Unendlichen. Zur Beschreibung der „Kosten“ eines Algorithmus wird in dieser Arbeit der Buchstabe ($\mathcal{O}$) verwendet. Dieser beschreibt, dass die betrachtete Funktion f höchstens so schnell wie $n \in \mathbb{N}_0$ im Fall von $f \in \mathcal{O}(n)$ wächst.

In den folgenden Beispielen ist $f = f(n)$ als Funktion von n zu verstehen.

Normalerweise ist es sehr aufwändig oder ganz unmöglich, für ein Problem L eine Funktion $f_L : w \to f_L(w)$ anzugeben, die allgemein jeder beliebigen Eingabe für ein Problem die zugehörige Anzahl der Rechenschritte (bzw. der Speicherzellen) zuordnet. Daher begnügt man sich in der Regel damit, statt

Tabelle 2.1: Beispiele für Kostenfunktionen

Notation	Bedeutung	Beispiele für Laufzeiten
$f \in \mathcal{O}(1)$	f ist beschränkt	Look Up in einer Hashtabelle
$f \in \mathcal{O}(\log n)$	f wächst logarithmisch	Binäre Suche im sortierten Feld mit n Einträgen
$f \in \mathcal{O}(n)$	f wächst linear	Suche im unsortierten Feld mit n Einträgen
$f \in \mathcal{O}(n \log n)$		Fortgeschrittene Sortieralgorithmen (z. B. Quicksort)
$f \in \mathcal{O}(n^2)$	f wächst quadratisch	Einfache Algorithmen zum Sortieren von n Zahlen.
$f \in \mathcal{O}(2^n)$	f wächst exponentiell	

jede Eingabe einzeln zu erfassen, sich lediglich auf die Eingabelänge $n = |w|$ zu beschränken. Es ist aber meist ebenfalls zu aufwändig, eine Funktion $f_L : n \to f_L(n), n = |w|$ anzugeben.

Aus diesem Grund wurde die Landau-Notation entwickelt, die sich auf das asymptotische Verhalten der Funktion f_L beschränkt. Man betrachtet daher, in welchen Schranken sich der Rechenaufwand (der Bedarf an Speicher und Rechenzeit) hält, wenn man die Eingabe vergrößert. Dabei ist die Funktion f nicht immer bekannt. Die Landau-Notation wird verwendet um den Rechenaufwand – oder den Platzbedarf – abzuschätzen wenn es zu aufwändig ist die genaue Funktion anzugeben bzw. wenn diese zu kompliziert ist.

Die Landau-Symbole erlauben es, Probleme und Algorithmen nach ihrer Komplexität in Klassen zusammenzufassen.

2.7 Formale Grammatiken

Formale Grammatiken sind mathematische Modelle die formale Sprachen erzeugen. Sie werden in der theoretischen Informatik, insbesondere in der Berechenbarkeitstheorie, und im Compilerbau angewandt. Eine formale Grammatik erlaubt es zu entscheiden, ob ein Text dieser Grammatik folgt, also ob er „gültig“ ist. Die Menge aller Texte, die der Grammatik folgen, nennt man die Sprache dieser Grammatik. Einen Text, der der Grammatik folgt, nennt man auch ein Wort dieser Sprache. Umgekehrt erlaubt es jede Grammatik, alle Wörter ihrer Sprache zu erzeugen.

Formale Grammatiken werden häufig als reguläre Ausdrücke angegeben. Die gängigen Notationen, welche weiter unten auch in der Backus-Naur-Form zu erkennen sind, verwenden Pseudosymbole, sogenannte Nichtterminalsymbole, die im Text nicht auftauchen. Oft werden hierfür Großbuchstaben verwendet. Diese Symbole wirken wie Platzhalter, die aufgrund von Regeln (sog. Produktionsregeln oder Produktionen) durch andere Platzhalter oder Terminalsymbole oder eine beliebige Kombination von Platzhaltern und Terminalsymbolen ersetzt werden. Die Regeln produzieren aus einer Folge von Terminalsymbolen und Nichtterminalsymbolen andere Folgen.

2.7.1 (E)BNF

Die Backus-Naur-Form (BNF) ist eine kompakte formale Metasyntax, die benutzt wird, um kontextfreie Grammatiken (= formale Grammatiken des Typs-2) darzustellen. Hierzu zählt die Syntax gängiger höherer Programmiersprachen. Die BNF wird auch für die Notation von Befehlssätzen und Kommunikationsprotokollen verwendet.

Ursprünglich war diese Form nach John Backus benannt, später wurde sie – auf Anregung von Donald E. Knuth – auch nach Peter Naur benannt [26]. Beide sind Informatikpioniere, die sich mit der Erstellung der Algol-60-Regeln [2] und insbesondere mit der Kunst des Compilerbaus beschäftigten. Durch die Backus-Naur-Form im Algol 60 Report wurde es erstmals möglich, die Syntax einer Programmiersprache formal exakt, also ohne die Ungenauigkeiten natürlicher Sprachen, darzustellen.

Es gibt viele Varianten der Backus-Naur-Form. Die erweiterte Backus-Naur-Form (EBNF) ist eine gebräuchliche Variante, die unter anderem eine kompakte Notation von sich wiederholenden Elementen erlaubt [48]. Dennoch lassen sich mit beiden Formen die selben Grammatiken wiedergeben.

Da die EBNF-Beschreibungen der alten und neuen Wissensbasen des weiteren in dieser Arbeit dargestellt und verglichen werden, soll näher auf die Eigenschaften und Regeln der EBNF eingegangen, und diese erklärt, werden.

Produktionen
Eine EBNF besteht aus einer Liste von Produktionen. Jede Produktion be-

schreibt die Syntax eines bestimmten Grammatikfragmentes. Produktionen werden als eine Art Gleichungen geschrieben. Auf der linken Seite steht ein Name für das definierte Grammatikfragment; auf der rechten Seite steht eine Folge von Symbolen, die den Aufbau des Grammatikfragmentes festlegen. Zwischen linker und rechter Seite wird – je nach Verfasser – das Trennzeichen `::=` oder `:=` gesetzt. Das Ende einer Produktion wird mit einem Punkt gekennzeichnet.

Schematisch sieht jede EBNF wie die nächste Zeile aus:

```
linke Seite ::= rechte Seite.
```

Terminale und Nichtterminale
Von den Symbolen, die auf der rechten Seite einer Produktion vorkommen, gibt es zwei Arten:

- Die Terminale stehen für sich selbst, sie sind wörtlich zu nehmen.
- Die Nichtterminale benennen eine andere Produktion, die an dieser Stelle einzusetzen ist.

Zur Unterscheidung werden Terminale oft in Anführungszeichen gesetzt, Nichtterminale unterstrichen, in Grossbuchstaben geschrieben, oder einfach überhaupt nicht markiert.

Symbolfolgen
Um eine Reihe von Symbolen auf der rechten Seite einer Produktion als optional zu kennzeichnen, wird diese in eckige Klammern gesetzt. Eine Symbolfolge wird hingegen in geschweifte Klammern gesetzt, wenn sie beliebig oft wiederholt werden darf. Das schließt Weglassen, d.h. null-maliges Wiederholen, mit ein. Das Beispiel für Syntaxdiagramme als EBNF:

```
integer ::= [sign] digit {digit}.
```

In diesem Zusammenhang ist eine alternative Notation üblich, die in der EBNF eigentlich nicht vorgesehen ist: Hinter die geschweifte Klammer kann auch ein Plus-Zeichen gesetzt werden, um ein- oder mehrmalige Wiederholung auszudrücken. Das obige Beispiel würde dann kürzer lauten:

```
integer ::= [sign] {digit}+.
```

Um die ursprüngliche, beliebige Wiederholung davon klar abzuheben, wird

diese dann ausdrücklich mit einem Stern nach der geschweiften Klammer markiert. Die folgende Schreibweise ist umständlicher, aber inhaltlich gleichwertig mit der vorhergehenden:

```
integer ::= [sign] digit {digit}*.
```

Alternativen
Wenn mehrere Symbolfolgen auf der rechten Seite einer Produktion zur Auswahl stehen, werden die Möglichkeiten nacheinander aufgeführt und mit senkrechten Strichen getrennt. Das folgende Beispiel zeigt die komplette EBNF für eine `integer`-Konstante in „C“-Syntax:

```
integer ::= [sign] digit {digit}.
sign ::= "+" | "-".
digit ::= "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9".
```

Gruppierung
Symbolfolgen können mit einfachen runden Klammern gruppiert werden, um Alternativen einzugrenzen. Auch die anderen Klammer-Arten (eckige und geschweifte) können so benutzt werden. Das folgende Beispiel ist zwar ungeschickt, illustriert aber diese Möglichkeit:

```
integer ::= [sign] ("0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9") {di-
git}.
sign ::= "+" | "-".
digit ::= "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9".
```

2.7.2 XML

Zum Verständnis der XML-Grammatik der alten Wissensbasis von RHEUMexpert/Web – welche in späteren Kapiteln zur Gegenüberstellung mit der neuen Lua-Form verwendet wird – ist die Kenntnis des Aufbaus von XML notwendig. Deswegen wollen wir im Folgenden näher auf die Eigenschaften von XML eingehen.

Die Ursprünge von XML liegen in der Standard Generalized Markup Language (SGML). Die Sprache wurde von Dr. Charles Goldfarb in den 70er Jahren entwickelt und findet breite Anwendung in High-End-Publishing-

Systemen [4]. Leider verhindert die Komplexität von SGML seine völlige Akzeptanz in der Industrie (SGML wird auch gerne mit „sounds great, maybe later“ übersetzt). SGML erlangte einen neuerlichen Aufschwung als Tim Berners-Lee in den späten 80er Jahren HTML auf SGML aufbaute und in der Folge das World Wide Web (WWW) begründete [42]. Seither benutzte die gesamte Computerindustrie diese Auszeichnungssprache zum Erstellen von Dokumenten und Anwendungen.

Der Gedanke hinter XML besteht darin, dass der Inhalt von der Präsentation getrennt wird. Daten/Regel-XML-Dokumente beschreiben deutlich die betreffenden Teile und bringen diese in ein Format, welches sich leichter parsen lässt, ohne aber Aussagen über deren Design oder deren grammatikalisches Format zu tätigen. Hierfür werden Dateien mit Dokumentenregeln oder andere Dateien mit Stylevorschriften vom entsprechenden XML-Dokument referenziert, um die Daten in einer gültigen Form in einem gewünschten Design darzustellen.

Für XML-Dokumente gibt es folgende Regeln die in [47] beschrieben sind:

Struktur / Schachtelung
Ein XML-Dokument muss in einem einzigen Element enthalten sein. Das erste Element in unserem XML-Dokument muss das gesamte Dokument enthalten. Dieses erste Element wird Root- oder Wurzelelement genannt. Enthält das Dokument mehr als ein solches Element, dann kommt es zu einem Ausnahmefehler, welchen der XML-Parser anzeigt (Exception).

Alle Elemente müssen geschachtelt sein. Wenn ein Element innerhalb eines anderen beginnt, muss es auch dort geschlossen werden. HTML-Browser halten sich an diese Vorschrift im Sinne einer Fehlertoleranz beim Parsen von Webseiten beispielsweise nicht.

Attribute
Alle Attribute müssen in Anführungszeichen stehen. Die Werte von Attributen – jene Schlüssel innerhalb der spitzen Klammern eines XML-Tags – müssen in einfache oder doppelte Anführungszeichen gesetzt werden. XML erlaubt weiters keine Attribute ohne Werte. Elemente wie `<td nospace>` sind in XML verboten und werfen eine Exception des Parsers. Eine korrekte Schreibweise des oberen Elements wäre `<td nospace="yes">`.

Tags

Tags sind Zeichenfolgen, die die Bedeutung der beinhalteten Informationen angeben. XML-Tags unterscheiden immer zwischen Gross- und Kleinschreibung. Es sind immer End-Tags erforderlich, welche die Informationsfolge, die mit dem Start-Tag begann, beenden. Dies sieht wie folgt aus:

```
<person vorname="'karl"'>weitere Information</person>
```

Man erkennt in obigem Beispiel, dass der `person`-Tag ein Attribut mit dem Namen `vorname` hat, welches den Wert `karl` trägt. Der Tag kann noch weitere Informationen beinhalten (ebenfalls auch weitere Tags) und wird mit `</person>` geschlossen.

Dies ist ein Bereich, in dem die meisten HTML-Dokumente versagen. Web-Browser überspringen Tags von Zeilenumbrüchen (`br`eak-Tags), für welche es keinen geschlossenen `</br>`-Tag gibt, XML-Parser hingegen nicht.

XML-Deklarationen

Viele XML-Dokumente beginnen mit einer XML-Deklaration. Diese ist eine Zeile, die wie folgt aussehen kann:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
```

Ist die Zeichencodierung nicht angegeben, interpretieren XML-Parser, dass das Unicode Transformation Format (UTF-8) – ein Unicode-Standard, der unterschiedliche Bytelängen bei der Zeichendarstellung gebraucht – verwendet wird. In deutschsprachigen Dokumenten wird UTF hingegen nicht gerne verwendet, da dieser Zeichensatz von einigen Editoren deutsche Umlaute falsch darstellt. Unterschiedliche XML-Parser interpretieren standardmässig teils unterschiedliche Zeichensätze, wodurch die Angabe der XML-Version und des Zeichensatzes immer grössere Verbreitung findet.

Nähere Informationen über XML können in [47] gefunden werden.

2.7.3 EBNF der alten Wissensbasis

Die von Reinhard Pitsch in seiner Magisterarbeit „RHEUMexpert-II-Web“ [39] verwendete Grammatik stützt er auf folgende EBNF Notation. Hierin

sind aber in Bezug auf eine komplette EBNF – mit welcher man die bestehende Wissensbasis auch lexen könnte – Fehler enthalten, da die Definition der entstehenden XML-Struktur nicht in der EBNF gegeben wird.

```
RULE := IF THEN ELSE?
IF   := EXPRESSION
THEN := ASSERTION+
ELSE := ASSERTION+
EXPRESSION := NEG EXPRESSION | COP EXPRESSION+ | LOP EXPRESSION EXPRESSION
              | VOP (SDE|ISDE|SYM|MATH) VAL
MATH := ABS? MOP SDE SDE
ASSERTION := AOP
AOP  := (ISDE|SYM) VAL
VAL  := ("true"|"false"|Integer-Zahl|Real-Zahl)
VOP  := ("<"|"≤"|"=="|"≥"|">"|"!=")
LOP  := ("and"|"or")
COP  := (atLeastInclude|atLeastExclude) (atMostInclude|atMostExclude)?
        (atMostInclude|atMostExclude)
MOP  := ("+"|"-")
NEG  := ("!")
ABS  := ("abs")
AOP  := ("=")
SDE  := ... enthält Attribute wie Index, Kategorie, Name, etc.
ISDE := ... enthält Attribute wie Index, Kategorie, Name, etc.
SYM  := ... enthält Attribute wie Index, Kategorie, Name, etc.
```

Diese Form dient zwar dem Verständnis des versierten Lesers, ist aber von einem Lexer nicht abzuarbeiten, sondern nur als Gedächtnisstütze von Pitsch [39] anzusehen.

Betrachtet man nämlich die erste Zeile, sieht man, dass eine Regel aus mindestens einer `EXPRESSION` und einer `ASSERTION` bestehen müsste, nicht aber aus dem Teil der das XML-`<IF>` verlangt. Hierbei ist die komplette XML-Struktur, wie auch z. B. die Indexposition der Regeln, außer acht gelassen, da die Regeln in der ehemals fertig gestellten Version wie in Listing 2.1 aussehen.

Listing 2.1: Darstellung der ersten Regel zur Veranschaulichung der EBNF

```
<RULE index="1" visible="-1">
  <IF>
    <EXPRESSION>
      <VOP value="==">
        <ISDE index="1"/>
        <VAL value="1"/>
      </VOP>
    </EXPRESSION>
  </IF>
  <THEN>
    <ASSERTION>
      <AOP value="=">
        <SYM index="1"/>
        <VAL value="1"/>
      </AOP>
    </ASSERTION>
  </THEN>
</RULE>
```

Wenn man die Sprache, welche durch eine korrigierte Version der obigen EBNF definiert wird, in die Form der XML-Regeln aus dem Listing 2.1 bringt, so erhält man die alte Wissensbasis von RHEUMexpert/Web – welche in [39, Anhang F, Seite 128ff.] aufgelistet ist.

Es soll betont werden, dass XML ideal (und richtungsweisend) für viele Anwendungsgebiete ist. Leider ist es aber nicht der beste Weg eine regelbasierte Wissensbasis aufzubauen. Man erkennt ab dem Kapitel 3 weswegen die Form der neuen Darstellung sowohl für Menschen, als auch für die Maschine geeigneter ist.

2.8 Softwarepakete

In diesem Kapitel sollen die Programme vorgestellt werden, welche für die Inhalte und Ergebnisse der vorliegenden Arbeit notwendig waren. Die Beschreibung umfasst die verwendeten Programmiersprachen, nicht aber eine Beschreibung der jeweiligen Editoren oder Benutzerinterfaces.

2.8.1 Lua

Lua (portugiesisch für Mond) ist eine Skriptsprache zur Einbindung in Programme, um diese leichter weiterentwickeln und warten zu können. Eine der besonderen Eigenschaften von Lua ist die geringe Größe des kompilierten Skript-Interpreters. Lua wurde 1993 von der Computer Graphics Technology Group der Pontifikalen Katholischen Universität von Rio de Janeiro in Brasilien entwickelt [5]. Lua ist freie Software, und wurde bis zur Version 5 unter einer BSD-, ab Version 5 unter der MIT-Lizenz veröffentlicht [45].

Die vorliegende Arbeit ist keinesfalls die erste, die sich mit der Skriptingsprache Lua beschäftigt. Lua wurde bereits in zahlreichen großen Softwareprojekten wegen seiner Stabilität und Flexibilität eingesetzt.

Anbei eine kleine Liste erfolgreicher Projekte [46]:

- Universitär:
  - GUPPY (AIST Tokyo)
  - Ribosome Builder (University of Montana)
  - noweb (Harvard University)
  - CARA (Kurt Wüthrich, Chemienobelpreis 2002, ETH Zürich), ...
- Industrielle Applikationen: Robotik, Bildverarbeitung, Bladecluster, ...
- Hardwareseitig: Embedded Systems, Ethernet switches, ...
- Computerspiele: WorldsOfWarcraft, Half-Life 2, Spellforce, FarCry, ...

Weitergehende Informationen zu Lua sind in [45] zu finden.

Einige Eigenschaften von Lua

- Interpretierte Skriptsprache,
- Effizienz/Typenlose Variablen,
- Ablauf in einer virtuellen Maschine,
- Portabilität/Plattformunabhängig,
- Frei verfügbarer Quellcode,
- Universell einsetzbar, OOP-fähig, beliebige Erweiterbarkeit,
- Komplette Sprache ist schon vorhanden.

2.8.2 LuaJava

Der LuaJava-Interpreter – welcher seit dieser Arbeit in seiner Version 1.1 in RHEUMexpert/Web zum Einsatz kommt – kann über eine Java-Bibliothek angesprochen werden, die auch eine Programmierschnittstelle (engl. Application Programming Interface oder kurz API) für die Laufzeitumgebung des Interpreters für Aufrufe vom Java-Programm aus beinhaltet. Mittels der API können verschiedene Teile des Programmes in Java und Lua geschrieben werden, während Variablen und Funktionen in beiden Richtungen erreichbar bleiben (d.h. eine Funktion in Lua kann eine Funktion in Java aufrufen, und umgekehrt).

Dennoch ist LuaJava effizient genug, um keinerlei Variablentypen in Lua verwenden zu müssen. Durch die Verwendung von LuaJava ist eine klare Trennung sämtlicher Bestandteile der Inferenz vom restlichen Sourcecode möglich. Weitere Informationen können in [44] gefunden werden.

2.8.3 PHP

PHP (rekursives Akronym für „PHP Hypertext Preprocessor“) ist eine Skriptsprache mit einer an C bzw. Perl angelehnten Syntax, die hauptsächlich zur Erstellung dynamischer Webseiten verwendet wird. Bei PHP handelt es sich um Open-Source-Software.

Es ist eine serverseitig interpretierte Sprache. Dies bedeutet, dass sie – im Gegensatz zu HTML oder JavaScript – den Quelltext nicht direkt an den Browser übermittelt, sondern ihn erst vom Interpreter auf dem Webserver ausführen lässt (Näheres zu Interpretern in Kapitel 2.4.2). Die Ausgabe der Applikation wird dann an den Browser geschickt. Das Ergebnis ist in den meisten Fällen eine HTML-Seite. Es ist aber auch möglich mit PHP andere Datentypen wie z. B. Bilder oder PDF-Dateien zu generieren oder es für Parsingprozesse und Skripting auf einem lokalen Rechner zu verwenden. Dies wurde im Zuge dieser Magisterarbeit oftmals getan.

PHP zeichnet sich besonders durch die geringen Applikationserstellungszeiten (Prototyping), die breite Datenbankunterstützung und Einbindung sämtlicher Protokolle die im Internet Anwendung finden, sowie die Verfügbarkeit zahlreicher zusätzlicher Funktionsbibliotheken – wie das XMLLib – aus, welche vor allem für die Überführung der alten XML-Repräsentation in die neue Lua-Form Verwendung fanden.

2.9 Vorangegangene Arbeiten

In diesem Kapitel werden die vorangegangenen Arbeiten zu der Thematik von RHEUMexpert genannt. Die verwendeten Begriffe stimmen mit den Begriffen aus den jeweiligen Arbeiten überein und erfordern auf Grund einiger Unterschiede zwischen den Arbeiten eine kurze Gegenüberstellung, welche in Tabelle 2.2 einzusehen ist.

Tabelle 2.2: Synonyme in verschiedenen RHEUMexpert-Versionen

RHEUMexpert-I	RheumaNet	RHEUMexpert-II bzw. -Web
Symptome	Symptome	Einzeldatenelemente
Symptomgruppen	Symptomkombinationen	Symptome
Verdachtsdiagnosen	Diagnosen	Verdachts- und Ausschlussdiagnosen

2.9.1 RHEUMexpert-I

Die erste Version von RHEUMexpert wurde 1998 vom Institut für Medizinische Computerwissenschaften an der Medizinischen Fakultät der Universität Wien in Zusammenarbeit mit der Firma Software Unlimited GmbH entwickelt [23, 27, 28]. Zum gegenwärtigen Zeitpunkt liegt keine das Programm bzw. den Entwicklungsprozess betreffende Dokumentation vor. Ebenso wurde nicht dokumentiert nach welchen Kriterien die Wissensbasis erstellt wurde.

Das Programm selbst wurde unter Zuhilfenahme der MFC Libraries mit Visual C++ programmiert und ist unter Windows95 und höher lauffähig. Das Programm bietet eine deutschsprachige graphische Benutzeroberfläche. Die Oberfläche wurde so gestaltet, dass die einzelnen Eingabemasken über so genannte Tabs (oft auch „Reiter“ genannt), aufrufbar sind.

Die Eingabedaten umfassen 615 Symptome, welche auf 71 Symptomgruppen abgebildet werden. Diese werden wiederum auf sieben Hauptdiagnosen mit 13 Unterdiagnosen abgebildet. RHEUMexpert-I deckt hierbei allerdings bei weitem nicht alle bekannten Diagnosen ab.

2.9.2 RheumaNet

Das Expertensystem RheumaNet wurde im Jahr 1999 von Ramin Gharakhanzadeh im Rahmen einer Diplomarbeit [36] entwickelt. Die Idee dahinter war ein webbasiertes Konsultationssystem in Java zu entwickeln. Als Datenbackend wurde die objektorientierte Datenbank POET verwendet. Das gesamte System wurde im Medframe Framework implementiert [37], welches die Entwicklung medizinischer Expertensysteme unterstützen sollte.

Zum gegenwärtigen Zeitpunkt sind weder Quellcodes noch Binärdateien von RheumaNet vorhanden. Reinhard Pitsch weist in seiner Magisterarbeit darauf hin, dass RheumaNet möglicherweise in Medframe integriert wurde und deshalb nicht mehr eigenständig existiert [39].

2.9.3 RHEUMexpert-II

Im Jahr 2000 wurde im Rahmen der Diplomarbeit „RHEUMexpert-II: Ein Dokumentations- und differentialdiagnostisches Konsultationssystem“ [40] das Expertensystem RHEUMexpert-II von Irfan Skiljan entwickelt. Es handelte sich hierbei aber um keine Neuentwicklung, sondern um eine Weiterentwicklung von RHEUMexpert-I. In diesem Sinn gibt es entsprechend viele Ähnlichkeiten zwischen beiden Programmen. Es wurde, so wie im Fall von RHEUMexpert-I ebenfalls Microsoft Visual C++ zur Umsetzung des Programmes verwendet. Die Benutzerschnittstelle ist ebenso wie die der Vorgängerversion aufgebaut. Die Wissensbasis wurde leicht verändert, RHEUMexpert-II umfasst 76 statt 71 Symptomgruppen.

2.9.4 RHEUMexpert/Web

Die aktuelle Version von RHEUMexpert wurde von Reinhard Pitsch im Rahmen seiner Magisterarbeit [39] nach Java portiert. Er verwendet in dieser die selben Definitionen/Synonyme wie in RHEUMexpert-II (siehe Tabelle 2.2). Das Projekt hiess anfangs noch RHEUMexpert-II-Web, wurde jedoch im Juli 2006 in RHEUMexpert/Web umbenannt.

Die Grundstruktur des Expertensystems wurde von der bisherigen Standalone-Architektur in eine Client-Server-Architektur [39, Seite 57ff.] geändert. Ziel der Arbeit war es ein webbasiertes – somit in einem Webbrowser aufrufbares – Expertensystem zu schaffen, welches einfachen Zugriff von beliebigen Betriebssystemen und Computerarchitekturen ermöglicht.

Die Arbeit von Reinhard Pitsch hatte die Implementierung der Server- und Evaluierungskomponente zum Ziel. Hierin wurde auch die Wissensbasis erstmals ausgelagert und in eine Grammatik übersetzt, welche mit ihrem Aufbau dem XML-Standard entspricht. Durch diesen Schritt wurde zum ersten Mal eine in Waterman [17] geforderte Trennung der Wissensbasis von der restlichen Programmverarbeitung ermöglicht.

2.10 Schnittstellen zu aktuellen Arbeiten

In diesem Kapitel werden die aktuellsten Arbeiten zum Thema RHEUMexpert angeführt. Die erste beschäftigt sich mit einem graphischen Editor für die Wissensbasis von RHEUMexpert. Die zweite befindet sich gerade in der Entstehung, und sollte im Jahre 2008 publiziert werden.

2.10.1 Wissenserwerbssystem für RHEUMexpert/Web

Die im Jahre 2006 fertiggestellte Magisterarbeit „Wissenserwerbssystem für RHEUMexpert/Web“ von Udo Kronberger [38] stellt eine Ergänzung zu der vorliegenden Arbeit dar. Kronberger erstellte darin einen graphischen Editor, der die neuen Wissensverarbeitungssprache – die in der vorliegenden Arbeit definiert wird – benutzt, um die Regeln und Sicherheitsfaktoren, die RHEUMexpert/Web zugrunde liegen, auch von Nichtinformatikern editieren lassen zu können.

Weiters stellt Kronberger in seiner Arbeit eine Erklärungskomponente vor, welche ebenfalls mit der vorliegenden Wissensverarbeitungssprache einhergeht. Durch diese ist es möglich sich zu einzelnen Patienten sämtliche Regelbäume anzusehen, die zum Ergebnis der ermittelten Verdachtsdiagnosen beigetragen haben.

2.10.2 Entwicklung und Wartung wissensbasierter Systeme

Die derzeit in Entstehung befindliche Doktorarbeit „Eine Architektur für die Entwicklung und Wartung wissensbasierter Systeme in der Medizin“ von Reinhard Pitsch befasst sich mit der Entwicklung einer Expertensystem-Shell für das System RHEUMexpert.

Die wesentlichen Ziele der Arbeit sind neben den theoretischen Aspekten insbesondere eine robuste und fehlerfreie Implementierung der Software bzw. der Softwarekomponenten sowie eine gut umgesetzte Benutzerschnittstelle, welche über das Internet – und damit mit jedem Webbrowser – verwendbar ist.

Eckpfeiler der Arbeit sind die drei Komponenten Konsultation, Evaluierung und Wissenserwerb. Die Arbeit baut dabei besonders im Fall der Konsultation auf die neuartige Inferenz auf, die durch die vorliegende Arbeit eingeführt wird. Weiters ist die nachhaltige Aufbereitung vorhandener Testdatensätze aus dem ehemaligen WAMIS (Abk. für Wiener Allgemeines Medizinisches Informationssystem) des AKH Wien, welche für das Evaluierungssystem Verwendung finden, ein wichtiges Ziel.

Neben der Softwareentwicklung hat diese Dissertation auch die Evaluierung und Überarbeitung der Wissensbasis zum Ziel. Dabei werden bestehende Inferenzmechanismen kritisch hinterfragt und diese auch anderen Methoden zum Vergleich gegenübergestellt. Im Zusammenhang mit dieser kritischen Hinterfragung ist es wichtig die verwendeten Algorithmen schnell (ohne lästiges Kompilieren) abzuändern, oder unterschiedliche Algorithmen auszuwählen. Hierdurch kommt der neuen Wissens-Engine besondere Bedeutung zu.

Jede der später erstellten Komponenten – aus Pitschs Arbeit – greift im Fall der Wissensabarbeitung auf die neue serverseitige Inferenz zu, welche in der vorliegenden Arbeit definiert und erstellt wird.

3 Methoden / Anwendung

In diesem Kapitel werden Gründe für die Wahl der Skripting-Sprache Lua gegeben, und anhand dieser erklärt, wie die einzelnen Schritte der Inferenz funktionieren und zusammenarbeiten.

RHEUMexpert/Web verwendet bis zu dieser Arbeit eine eigene Inferenzmaschine, die zuerst die XML-Regeln in eine Java-Klassenstruktur bringt, und die Eingaben des Benutzers dann mit diesen Java-Klassen abarbeitet [39, Seiten 65-69]. Um die XML-Regeln nicht jedes Mal neu einzulesen (und zu interpretieren), verwendet Pitsch ein Cachingsystem [39, Seite 72], welches die Abarbeitung zwar ein wenig performanter macht, aber dafür mehr Arbeitsspeicher benötigt.

Wenn (wie im Fall der Regel 31 im zweiten Schritt der Inferenz) Hunderte dieser Regelklassen befüllt, und dann abgearbeitet werden müssen, sieht man die Grenzen dieser Methode. Da die Inferenz-Regel-Funktionen tatsächlicher (vereinfachter) Lua-Quellcode sind, der direkt von LuaJava abgearbeitet wird, ist die Abarbeitungsgeschwindigkeit (und damit auch die Serverlast) mit der neuen Methode wesentlich geringer.

Ähnlich wie Shortliffe und Buchanan [3, Seite 6f.] wurde die Entscheidung getroffen, dass eine mächtige, dennoch vielseitig einsetzbare Programmiersprache für die Regeln – und Abarbeitung – der Wissensbasis von großem Nutzen ist. Im Fall von MYCIN wurde vor mehr als 30 Jahren die Sprache LISP aufgrund ähnlicher Eigenschaften gewählt, wegen der Lua für die vorliegende Arbeit Anwendung findet. Selbstverständlich gibt es enorme rechen- und softwaretechnische Fortschritte, dennoch sind Eigenschaften wie Flexibilität, gute Verwendbarkeit für rapid Prototyping und die Interpretation der Wissensbasis zur Laufzeit noch genauso wichtig wie dies vor 30 Jahren der Fall war.

Nachdem die Entscheidung für die Sprache getroffen war, musste eine geeignete Syntax für die umzuwandelnde Wissensbasis gefunden werden. Hierbei war vor allem für die graphische Repräsentation der Sprache eine von Schleifen und Zuweisungen freie Wissensrepräsentation nötig.

Eine interessante Herausforderung war es, den eigentlichen Inferenzablauf

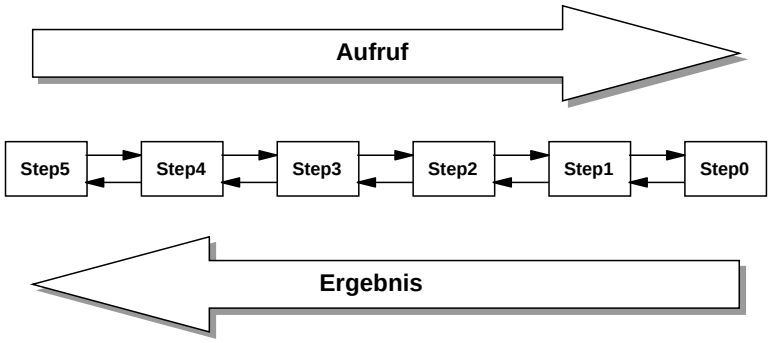

Abbildung 3.1: Vereinfachte Darstellung der Inferenz

– also wie mit Hilfe der Wissensbasis und der Eingabedaten neues Wissen abgeleitet werden kann – zu bestimmen. Die eigentlichen Abläufe in der Inferenzmaschine stimmen zum aktuellen Zeitpunkt mit denen der alten Wissensverarbeitung von RHEUMexpert/Web überein. Die neue Grammatik ist jedoch in einer Form, welche sich wesentlich besser zur graphischen Darstellung und zur Hinzufügung von Regeln eignet.

Die Idee hinter der neuen Inferenz war es den letzten Schritt der Inferenz nach dem Ergebnis zu fragen, welcher dann den vierten, dieser den dritten usw. Schritt der Inferenzmaschine aufruft, und die Ergebnisse des jeweils vorigen Schrittes an den aktuellen zurück gibt. In Abbildung 3.1 ist der Lua-seitige Programmablauf schematisch dargestellt. Diese Graphik zeigt eine äusserst simplifizierte Version der Wissensverarbeitung, welche dennoch dem Verständnis hilft.

Es war der Wunsch, jeden Schritt atomar zu sehen um alle weiteren und vorigen Schritte wie Blackboxes behandeln zu können. Durch die schrittweisen Inferenz-Aufrufe der Lua-Regeln war dies beinahe uneingeschränkt möglich. Der dritte Schritt der Abarbeitung ist eine Ausnahme, die sich durch die Funktionsaufrufe ergibt, welche die interne Matrix-Struktur der Lua-Regeln darstellt.

Diese Art der Verarbeitung hat beinahe die Eleganz von rekursiven Aufrufen in anderen Projekten. Bevor näher auf die einzelnen Schritte eingegangen wird, soll die exaktere Abbildung 3.2 dem Verständnis des Kapitels helfen.

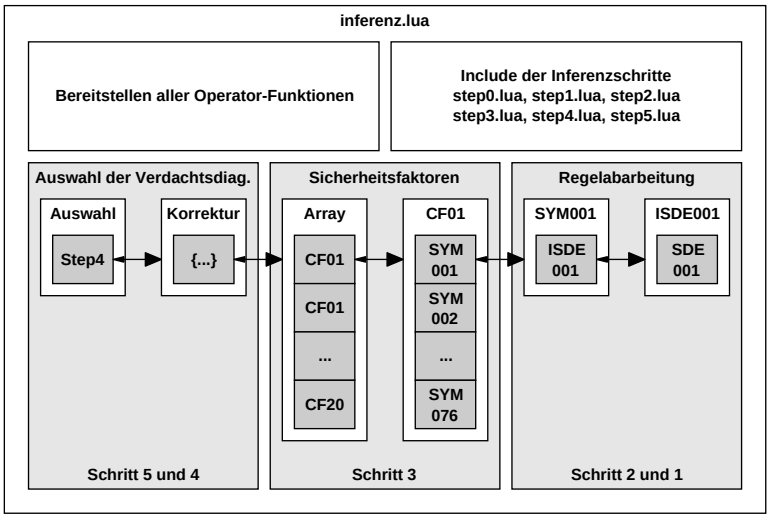

Abbildung 3.2: Komplexere Darstellung der Inferenz

3.1 Definition der neuen Grammatik

Die Grammatik der Sprache Lua – welche schon in Kapitel 2.8.1 näher beschrieben wurde – ist eine Übermenge der Grammatik welche für die RHEUMexpert/Web-Regeln definiert wurde. Aufgrund des graphischen Editors von Kronberger waren gewisse Eigenschaften und Fähigkeiten von Lua für die neue Wissenssprache ausgeschlossen.

Folgende Liste gibt einen Überblick der Forderungen an die neue Sprache:

- keine Variablen,
- keine mittels `if` erstellten Abfragen,
- keine Schleifenkonstrukte (`for`, `while`, `do`, ...),
- Abkapselung der einzelnen Regeln vom restlichen Quelltext,
- direkte Parameterübergabe,
- Box-in-Box-in-Box-Prinzip der Regeln,
- alleinige Verwendung eigener Operatoren und den
- Aufruf einer einzigen Funktion um das komplette Ergebnis zu erhalten.

Bevor wir auf die EBNF der Sprache eingehen, welche all diese Forderungen erfüllt, soll die EBNF von Lua – der Übersprache – angeführt werden:

```
chunk     := {stat [`;´]}
block     := chunk
stat      := varlist1 `=´ explist1 | if exp then block [else block] end
varlist1  := var {`,´ var}
var       := Name | prefixexp `[´ exp `]´ | prefixexp `.´ Name
init      := `=´ explist1
explist1  := {exp `,´} exp
exp       := nil | false | true | Number | prefixexp | tableconstructor
             | exp binop exp | unop exp
prefixexp := var | `(´ exp `)´
tableconstructor := `{´ [fieldlist] `}´
fieldlist := field {fieldsep field} [fieldsep]
field     := `[´ exp `]´ `=´ exp | name `=´ exp | exp
fieldsep  := `,´ | `;´
binop     := `+´ | `-´ | `<´ | `<=´ | `>´ | `>=´ | `==´ | `~=´ | and | ..
unop      := `-´ | not
```

Anbei nun die Definition der neuen Sprache:

```
step     := rule {rule}
rule     := "function" name "(" [paramlist] ")" body "end"
paramlist:= "(" variablename {"," variablename} ")"
body     := "return" operator
operator := opname "(" [oplist] ")" | value
opname   := "_sum" | "_gt" | "_lt" | "_ge" | "_between" | ...  -> Kapitel 3.3.1
oplist   := operator {"," operator}
```

Man erkennt die Einfachheit, mit welcher sämtliche Forderungen durch Funktionsaufrufe und Rückgabeparameter erfüllt werden. In den nächsten Kapiteln wird genauer auf die einzelnen Operatoren, Funktionsaufrufe und die Ideen hinter diesen eingegangen.

3.2 Umwandlung der XML-Regeln nach Lua

Nach der Spezifikation der Grammatik der Regeln, musste ein Programm geschrieben werden, welches die ersten drei Schritte der Wissensverarbeitung zu jedem beliebigen späteren Zeitpunkt aus der alten XML-Struktur in die neue Lua-Form überführt. Durch den hervorragenden XML-Support in PHP fiel die Entscheidung auf diese Sprache um das Konvertierprogramm darin zu realisieren.

Hierbei werden die jeweiligen Dateien der Inferenzschritte eins bis drei in die interne Klassenstruktur von PHP eingelesen, und dann die Pitsch'schen XML-Regeln [39, Anhang F, Seite 128ff.] in Lua-Code umgewandelt. Der entsprechende Programmcode umfasst weniger als 300 Zeilen und ist im Listing A.1 zu finden.

Die Schritte vier und fünf und Teile des Schrittes drei der Wissensverarbeitung waren in der alten Version von RHEUMexpert/Web im Sourcecode kompiliert und damit nicht zur Laufzeit änderbar. Dieses Problem wurde durch die Verwendung von Lua aufgehoben. Die Aufgabe war es bei der Inferenz (Schritte drei bis fünf) möglichst ohne Schleifenkonstrukte auszukommen. Hierzu wurde der Java-Quelltext von RHEUMexpert/Web analysiert und vom Autor der vorliegenden Arbeit – mittels eines PHP-Programmes welches in Anhang A.1 zu sehen ist – nach Lua übersetzt.

Zur Zeit der Fertigstellung dieser Arbeit existierten keine weiteren Arbeiten zur visuellen Abänderung der Schritte drei bis fünf. Dennoch wird es zur Laufzeit möglich sein, bei der Evaluation neuer Regeln/Datensätze, unterschiedliche vorgefertigte Algorithmen auszuwählen.

3.3 Generelle Funktionsbeschreibung

Im Folgenden werden die mathematischen Funktionen, welche zur Abarbeitung der Regeln eins bis drei in Lua erstellt und zur Verfügung gestellt wurden, aufgelistet. Die Namensgebung wurde vor Beginn des Entwurfes mit Kronberger abgeklärt [38, Seite 11] um gleichzeitig mit beiden Magisterarbeiten fortfahren zu können. Leider wurde das Problem der Sicherheitsfaktorenmatrix erst zu einem späteren Zeitpunkt aufgegriffen, wodurch

sich diese grundsätzlich änderte und deswegen eine leicht abgeänderte Namenskonvention hat.

Die Schritt-Funktionen der Inferenz haben vom ursprünglichen Aufbau in XML unterschiedlichste Anforderungen an ihre Beschreibungsform. Dennoch wurde eine Grammatik gefunden, die für die ersten Schritte der Inferenz ident ist. Das Grundkonzept wurde derart definiert, dass die Regeln eines Schrittes nur auf Ergebnisse des vorherigen Schrittes zugreifen können. Das angedachte Blackbox-Prinzip lässt sich durch diese Definition mit Lua-Funktionsaufrufen realisieren. Durch die neue Grammatik wird jede Regel durch eine Lua-Regel-Funktion abgebildet. Entsprechend den Vorgaben der Grammatik hat eine solche Regel genau einen Rückgabewert – welcher aber auch ein Array sein kann – und in den ersten drei Schritten der Inferenz keinen Aufrufparameter. Hierdurch wird schrittweise jede Regel-Funktion abgearbeitet und sie gibt den entsprechenden Booleschen Wert zurück. Schritt 4 und 5 sind Ausnahmen, da in diesen Fällen keine Regel-Funktionen mehr existieren, sondern Algorithmen auf die Gesamtheit der Ergebnisse angewendet werden.

Die javaseitigen Eingabedaten werden weiter im Schritt 0 beschrieben. Die Ergebnisse der Regeln eines Schrittes bilden wiederum Parameter der Funktionen im nächsten Schritt.

Listing 3.1: Der Aufbau einer Lua-Regel-Funktion

```
function funktionsname(...)
  return operatoren(parameter)
end
```

In Listing 3.1 ist `parameter` die Menge der nachfolgend beschriebenen Funktionen, die im Rahmen der Inferenz – ineinander verschachtelt – verwendet werden. Für jeden Schritt ist die Menge an Funktionen definiert, aus denen die Regeln des Schrittes aufgebaut werden. Diese Funktionen, welche 0 bis n Aufrufparameter aufweisen können, sind hier aufgelistet.

3.3.1 Operatorfunktionen der ersten Schritte

Folgende `functions` können in allen Schritten der Inferenz verwendet werden, sind aber die Bausteine aus denen die Regeln der Schritte eins und zwei zusammengesetzt sind:

- `_and(...)` Wenn alle übergebenen Variablen – dies sind sowohl Arrays als auch unbeschränkt viele Variablen – grösser als 0 sind (der Abschnitt 2.5.2 über dreiwertige Logik beschreibt diese Vorgehen näher), wird das Ergebnis dieses Operators gleich einem Wert der als `true` aufgefasst wird.
- `_or(...)` Wenn mindestens eine der übergebenen Variablen – dies sind sowohl Arrays als auch unbeschränkt viele Variablen – grösser als 0 ist (der Abschnitt 2.5.2 über dreiwertige Logik beschreibt diese Vorgehen näher), wird das Ergebnis dieses Operators gleich einem Wert der als `true` aufgefasst wird.
- `_gt(element1,element2)` Wenn die Realzahl `element1` grösser als die Realzahl `element2` ist, wird das Ergebnis von `_gt` gleich einem Wert, der als `true` auffassbar ist.
- `_ge(element1,element2)` Wenn die Realzahl `element1` grösser oder gleich der Realzahl `element2` ist, wird das Ergebnis von `_ge` gleich einem Wert, der als `true` auffassbar ist.
- `_lt(element1,element2)` Wenn die Realzahl `element1` kleiner als die Realzahl `element2` ist, wird das Ergebnis von `_lt` gleich einem Wert, der als `true` auffassbar ist.
- `_le(element1,element2)` Wenn die Realzahl `element1` kleiner oder gleich der Realzahl `element2` ist, wird das Ergebnis von `_le` gleich einem Wert, der als `true` auffassbar ist.
- `_equal(...)` Mit dieser Operator-Funktion kann man die Gleichheit zweier Variablen überprüfen. Wenn nur ein Parameter übergeben wird, so nimmt `_equal` an, dass dieser ein Array ist, welches mehrere Werte enthält, und prüft ob alle diese Elemente gleich dem ersten sind. Wenn zwei Arrays übergeben werden, so prüft die Funktion die Gleichheit der

Speicherbereiche in denen die Arrays gespeichert sind. Falls die Werte der beiden Arrays gleich (aber die Speicherbereiche unterschiedlich) sind wird `false` zurückgegeben. Wenn kein Parameter übergeben wird ist das Ergebnis ebenso als `false` auffassbar.

- `_eq(...)` Diese Funktion ist mit `_equal` ident.
- `_between(count,lbound,ubound[,greater_equal[,less_equal]])` Die Funktion benötigt ein Minimum von drei Übergabeparametern `count`, `lbound` und `ubound`). Hierbei wird geprüft, ob der erste Parameter grössergleich ($\geq$) dem zweiten Parameter und kleiner ($<$) als der dritte ist. Der zweite Parameter stellt somit die untere Grenze und der dritte respektive die obere Grenze des zu prüfenden Intervalls dar. Mit der Übergabe des vierten und fünften Parameters kann man bestimmen, ob die Grenzen mit kleiner- bzw grössergleich oder nur mit kleiner und grösser überprüft werden. Die Rückgabe von `between` kann als `true` aufgefasst werden, wenn der erste Parameter tatsächlich entsprechend den Anweisungen im Intervall der angegebenen Grenzen liegt.
- `_sum(elements)` Diese Funktion akzeptiert einen Parameter der ein Array mit beliebig vielen Zahlen ist, welche dann mit Hilfe der Funktion `_add` summiert werden. Die Rückgabe von `_sum` ist eine Realzahl.
- `_add(element1,element2)` Dies ist eine Funktion, welche zum Zweck einer späteren Fuzzifizierung eingefügt wurde, um später die Addition zweier Zahlen mithilfe einer speziellen Fuzzy-Funktion durchführen zu können. Zur Zeit der Fertigstellung dieser Arbeit, addiert `add` den ersten Übergabeparameter mit dem zweiten. Die Rückgabe ist demnach eine Realzahl.
- `_sub(element1,element2)` Die Funktion `_sub` dient so wie auch `_add` einer späteren Fuzzifizierung. Sie subtrahiert den zweiten Übergabeparameter vom ersten. Das Ergebnis ist eine Realzahl.
- `_mult(element1,element2)` Genau wie `_add` und `_sub` ist auch `_mult` zum Zweck einer späteren Fuzzifizierung im Funktionsumfang, da dann der Quellcode der Regel der Inferenz nicht geändert werden muss, wenn sich die Multiplikation zweier Elemente ändert.

- `_boundmult(element1, element2, lbound, ubound)` Dies ist die begrenzte Multiplikation zweier Elemente, welche keinen kleineren Wert als `lbound` und keinen grösseren als `ubound` ergibt. Hierbei wird zuerst die Funktion `_mult` aufgerufen um `element1` mit `element2` zu multiplizieren. Danach wird die jeweilige Grenze als Maximalwert zurückgegeben, falls diese überschritten ist. `_boundmult` findet zum Zeitpunkt der Fertigstellung dieser Arbeit nur im vierten Schritt – der Inferenz – Anwendung, da hierbei die korrigierten Sicherheitsfaktoren der Verdachtsdiagnosen nicht grösser oder kleiner ± 1 sein dürfen.
- `_abs(element)` Wenn eine Realzahl, oder ein String der als Zahl interpretierbar ist, an `_abs` übergeben wird, ist das Ergebnis der Absolutbetrag des Wertes. Die Rückgabe ist eine Realzahl.
- `_neg(element)` Wenn `_neg` ein logischer Wert übergeben wird, so ist das Ergebnis die Realdarstellung des invertierten logischen Wertes. `_neg(true)` wird `0`, `_neg(false)` wird `1`. Wenn eine Realzahl übergeben wird, so wird jeder Wert grösser 0 von der Funktion als `true` interpretiert, und das invertierte Ergebnis als Realzahl ausgegeben.

3.3.2 Ergänzende Funktionen

- `_valueize(element)` Mit dieser Funktion wird jeder übergebene Wert auf seine Darstellung in den Realzahlen umgewandelt. Wenn eine Realzahl übergeben wird, wird diese auf drei Kommastellen gerundet. Dies ist notwendig um die dreiwertige Logik im ersten Lua-Regel-Schritt zu realisieren. Strings werden auf deren Realzahl-Abbildung umzuwandeln versucht. Bei der Übergabe von `nil` wird so wie bei `false` die Zahl 0 zurückgeliefert.
- `_cfinner(sym,value)` Diese Funktion erhält als ersten Parameter ob ein gewisses Symptom existiert (`true` oder `false`) und multipliziert den Wahrheitswert des Symptoms mit dem Sicherheitsfaktor, der im zweiten Parameter übergeben wird. Mit Hilfe dieser Funktion wird es zu einem späteren Zeitpunkt möglich sein, Fuzzy-Logik in die Inferenz von RHEUMexpert/Web einzufügen.

- `_cfouter(...)` Mit dieser Funktion werden Ergebnisse von `cfinner` parallel aufsummiert. Um leicht neue Verdachtsdiagnosen hinzufügen zu können, ist es möglich beliebig viele Parameter zu übergeben. Durch eine Abänderung dieser Funktion können beliebige neue Aufsummierungsalgorithmen realisiert und getestet werden.

- `print_r(element[,indent,done])` Diese Funktion entspricht in der Verwendung ihrem Pendant in PHP. Sie zeigt Informationen über eine Variable in lesbarer Form an. Wenn die Funktion für eine Variable vom Typ string oder real aufgerufen wird, wird der Wert der Variablen angezeigt. Falls `print_r` für ein (mehrdimensionales) Array – welches in Lua den Typennamen `table` hat – aufgerufen wird, werden die Indizes und Werte des Arrays (rekursiv) angezeigt.

- `print_xml(element[,indent[,done]])` Das Verhalten der Funktion `print_xml` ist dem von `print_r` ähnlich, nur wird hier die Ausgabe nicht in einer lesbaren Form, sondern in XML zurückgegeben. Dies findet hauptsächlich im Fall der Datenrückgabe der Diagnosen an den Java-Server statt.

- `positives(...)` Alle übergebenen Elemente eines Arrays werden mit `positives` darauf überprüft, ob sie ≥ 0 sind. Die Elemente für die jene Bedingung zutrifft werden wieder als Array zurückgegeben.

- `negatives(...)` Genau wie bei `positives` wird hier eine Gruppe von Elementen zerteilt. Alle übergebenen Elemente eines Arrays werden darauf überprüft ob sie < 0 sind. Die Absolutbeträge, der Elemente für die jene Bedingung zutrifft, werden wieder als Array zurückgegeben.

- `sortIt(rray,hunni)` Hier ist ein einfacher *Minsort-* oder *Selectsortalgorithmus* implementiert. `sortIt` wird in allen Algorithmen des fünften Schrittes gebraucht, die nach Sicherheitsfaktoren geordnete Verdachtsdiagnosen verwenden. Da zur Zeit nicht mehr als 20 verschiedene Diagnosen zur Verfügung stehen, und es absehbar ist, dass es nie mehr als vielleicht 100 werden, stellen die Laufzeitkosten von $\mathcal{O}(n^2)$ kein Problem dar. Der entsprechend $\mathcal{O}(n)$ linear anwachsende Speicherverbrauch ist in der Serverumgebung erwünscht. Näheres kann in Kapitel 2.6 nachgelesen werden. Mit dem zweiten Parameter

kann man den Wertebereich der geordneten Elemente festlegen. Dies ist im Fall vom vereinfachten Otsu-Algorithmus des fünften Schrittes im Abschnitt 3.9.6 nötig, da die Varianz der Elemente im Bereich [0, 1] ungünstiger zu berechnen ist, als im Bereich [0, 100].

3.4 Schritt 0: Definitionen

Dieser Schritt wurde erstmals hinzugefügt, da er – obwohl für die Inferenz eigentlich unnötig – für das Verhalten des graphischen Editors von Kronberger unerlässlich ist. Falls weitere Eingangsvariablen in der Benutzeroberfläche von RHEUMexpert/Web hinzukommen, müssen diese auch dem graphischen Editor der Regeln bekannt gemacht werden. Diese Aufgabe erfüllt der Schritt 0. Hierin werden allen möglichen Eingangsvariablen (zur Zeit SDE001 bis SDE617) Initialwerte zugewiesen, falls die Variablen nicht schon vom aufrufenden Java-Server befüllt wurden.

Listing 3.2: Initialisierung der Eingabevariablen

SDE001=SDE001 **or** −0.00000001
SDE002=SDE002 **or** −0.00000001
−− *[...]*
SDE617=SDE617 **or** −0.00000001

Man erkennt im obigen Quellcode eine Eigenheit von Lua: hier wird der Variable `SDE001` ihr eigener Wert zugewiesen, falls die Variable schon initialisiert wurde. Falls nicht wird mit dem `or`-Zweig fortgesetzt, indem der Initialwert `-0,00000001` gesetzt wird [5]. Warum die Wahl auf gerade diesen Wert als Initialwert fiel, ist im Abschnitt 2.5.2 über dreiwertige Logik nachzulesen.

Kronbergers Editor verwendet diese Initialwerte um seine Klassenstruktur aufzubauen und die zu verwendeten Eingangsvariablen übergeben zu bekommen. Man kann den Editor auf diesem Weg aber auch mit den Daten einzelner Patienten befüllen, da man die entsprechenden Eingabevariablen einfach mit den Patientendaten als Standardwerte übergeben kann. Dies ist die einfachste Variante um den graphischen Editor als Erklärungskomponente zu verwenden.

Man kann in Listing 3.2 erkennen, dass sowohl die Eingangsvariablen, als auch sämtliche weitere Lua-Regel-Funktionen zwar der Namensgebung von Pitsch entsprechen, dies aber nicht zwingend der Fall ist. Eingangsvariablen könnten in den Lua-Regeln wesentlich sprechendere Namen erhalten, wodurch die Regeln sogar ohne dem graphischen Editor lesbar wären.

Der Schritt 0 wurde im Prototyping oftmals verwendet, da hier die Korrektheit der Regeln leicht ohne die LuaJava-Umgebung überprüfbar war. Hierfür konnte einfach der Kommandozeilen-Lua-Interpreter mit dem Inferenz-File geöffnet werden, wodurch das Ergebnis ohne Java-Server, Userinteraktion oder XML-Rückgabe auf der Benutzerkonsole angezeigt wurde. Hierdurch ist die Inferenz noch immer auch als Stand-alone-Programm betreibbar, auch wenn dies wohl kaum Anwendung finden wird.

3.5 Schritt 1: Interpretation der Eingaben

Dieser Schritt ist eine Abbildung der Einzeldatenelementsvariablen aus dem vorigen Schritt – bzw aus den Aufrufparametern der Inferenz aus Java – auf deren Boolesche Interpretation. Die Ausprägungen der Variablen, die dem Schritt überantwortet sind, werden mit den Regeln – welche in Lua-Funktionsaufrufen enthalten sind – an den nächsten Schritt zurück gegeben. Das Ergebnis der einzelnen Regel-Aufrufe besagt ob die Interpretation auftritt, oder nicht.

Anbei noch eine Gegenüberstellung, einer Regel des ersten Schrittes zwischen XML und Lua, welche einen Patienten aufgrund des Alters (`SDE617`) in die Klasse der 41- bis 70jährigen zuordnet.

Listing 3.3: Regel 625 des ersten Schrittes in der alten XML-Form

```
<RULE index=”625” visible=”−1”>
  <IF>
    <EXPRESSION>
      <LOP value=”and”>
        <EXPRESSION>
          <VOP value=”&gt;=”>
            <SDE index=”617”/>
```

```
            <VAL value="41"/>
          </VOP>
        </EXPRESSION>
        <EXPRESSION>
          <VOP value="&lt;=">
            <SDE index="617"/>
            <VAL value="69"/>
          </VOP>
        </EXPRESSION>
      </LOP>
    </EXPRESSION>
  </IF>
  <THEN>
    <ASSERTION>
      <AOP value="=">
        <ISDE index="625"/>
        <VAL value="1"/>
      </AOP>
    </ASSERTION>
  </THEN>
  <ELSE>
    <ASSERTION>
      <AOP value="=">
        <ISDE index="625"/>
        <VAL value="0"/>
      </AOP>
    </ASSERTION>
  </ELSE>
</RULE>
```

Listing 3.4: Regel 625 des ersten Schrittes in der neuen Lua-Form

```
function ISDE625 ()
  return _between(SDE617,41,70)
end
```

Anhand dieser (äusserst kurzen) Regel kann man folgern, wie schwer die

XML-Form von längeren Regeln zu lesen ist. Der `<THEN>`-Teil der `<IF>`-Abfrage in XML wird in Lua durch den Return-Wert ersetzt, der damit das interpretierte Einzeldatenelement 625 mit `true` oder `false` zurückgibt.

Funktionen die aktuell in Regeln des ersten Schrittes verwendet werden:

- `_abs` Absolutbetrag
- `_between` Zwischen
- `_eq` Gleichheit
- `_ge` Grössergleich
- `_gt` Grösser
- `_sub` Subtraktion

Die nähere Beschreibung dieser Funktionen befindet sich in Kapitel 3.3.1.

3.6 Schritt 2: Symptomfindung

Die im ersten Schritt ermittelten 626 (interpretierten) Einzeldatenelemente werden mithilfe der Regeln aus dem zweiten Schritt auf 76 Symptome abgebildet. Diese Lua-Symptom-Funktionen geben wieder `true` oder `false` an den nächsten Schritt der Inferenz zurück.

Einige der Regeln aus diesem Schritt sind selbst im Lua-Code mehrere Seiten lang. Die Länge dieser Regeln in der alten XML-Version [39, Anhang F, Seite 128ff.] kann man mit der Länge der neuen Regeln im Listing C.1 vergleichen. Um dennoch eine kleine Gegenüberstellung dieser Regeln zu geben, wird wieder eine äusserst kurze verwendet. Da aber selbst diese sehr viele Zeilen umfasst, wurde der `<IF>`-Teil um mehr als hundert `<EXPRESSION>`-Zeilen gekürzt.

Listing 3.5: Regel 36 des zweiten Schrittes in der alten XML-Form

```
<RULE index="36" visible="-1">
  <IF>
    <EXPRESSION>
      <NEG>
        <EXPRESSION>
```

```
        <COP atLeastInclude="1">
          <EXPRESSION>
            <VOP value="==">
              <ISDE index="37"/>
              <VAL value="1"/>
            </VOP>
          </EXPRESSION>
          <EXPRESSION>
            <VOP value="==">
              <ISDE index="38"/>
              <VAL value="1"/>
            </VOP>
          </EXPRESSION>
          [...]
          <EXPRESSION>
            <VOP value="==">
              <ISDE index="361"/>
              <VAL value="1"/>
            </VOP>
          </EXPRESSION>
        </COP>
      </EXPRESSION>
    </NEG>
  </EXPRESSION>
</IF>
<THEN>
  <ASSERTION>
    <AOP value="=">
      <SYM index="36"/>
      <VAL value="1"/>
    </AOP>
  </ASSERTION>
</THEN>
<ELSE>
  <ASSERTION>
    <AOP value="=">
```

```
        <SYM index="36"/>
        <VAL value="0"/>
      </AOP>
    </ASSERTION>
  </ELSE>
</RULE>
```

Listing 3.6: Regel 36 des zweiten Schrittes in der neuen Lua-Form

```
function SYM036 ()
  return _neg(_ge(_sum({ISDE037(),ISDE038(),..,ISDE361()}),1))
end
```

Zur Erklärung: Da `_sum` jeden Übergabeparameter als Zahl auffasst, wird `true` als 1 und `false` als 0 interpretiert. Hierdurch kann man die Abfrage ob mindestens eine der Regel-1-Funktionen „feuert“ mittels der Summenfunktion `_sum`, deren Wert im Symptom 36 grösser als 1 sein muss, überprüfen. Als letzter Aufruf, wird das Ergebnis der Summenfunktion noch negiert, wodurch für das Symptom 36 keines der Einzeldatenelemente auftreten darf.

Es ist informatisch zwar einfacher mit Indizes (z. B. der Symptome) zu arbeiten, dies entspricht aber nicht der Denkweise vieler anderer Menschen. Die neue Wissensverarbeitung bietet grundsätzlich die Möglichkeit, alle Lua-Regel-Funktionen beliebig – nicht nur mit deren Arten und Indizes – zu benennen.

Die Regel für das Symptom 36 muss nicht zwingend `SYM036()` heissen; sie könnte auch einen sprechenderen Namen wie z. B. `HWS_BWS_LWS_oB()` erhalten [39, Seiten 120–125]. Bei manchen Symptomen gibt es geeignetere Beschreibungen, welche ebenfalls verwendet werden könnten.

`SYM020()` vs. `radikulaere_Symptomatik()`
`SYM025()` vs. `Roetung_eines_oder_mehrerer_Gelenke()`
`SYM031()` vs. `entzuendl_Befall_eines_od_mehrerer_distaler_Gelenke()`
...

Funktionen die aktuell in Regeln des zweiten Schrittes verwendet werden:

- `_abs` Absolutbetrag
- `_and` Logisches Und
- `_between` Zwischen
- `_eq` Gleichheit
- `_ge` Grössergleich
- `_gt` Grösser
- `_neg` Negation
- `_or` Logisches Oder
- `_sub` Subtraktion
- `_sum` Summenfunktion

Die nähere Beschreibung der Arbeitsweise dieser Funktionen befindet sich in Kapitel 3.3.1.

Es sei vermerkt, dass Kronberger – der gleichzeitig an dem graphischen Editor dieser Regeln arbeitete – mit der Umsetzung der eingeführten Funktion `_between` im zweiten Schritt Probleme mit dem Aufbau seines Programmes hatte, wodurch `_between` nicht zur vollen Gänze Anwendung findet. Statt dessen, wurde die gleiche Funktionalität mit der Grösser- bzw. Kleinerfunktion erfüllt.

3.7 Schritt 3: Sicherheitsfaktoren

Nachdem die ermittelten Symptome aus dem zweiten Schritt bekannt sind, werden Sicherheitsfaktoren für jede Verdachtsdiagnose ermittelt. Hierbei wird eine Funktion zur parallelen Aufsummierung der Sicherheitsfaktoren verwendet, welche in [16] und in [3] näher beschrieben sind. Hierbei werden die Sicherheitsfaktoren zu jedem auftretenden Symptom einer Verdachtsdiagnose aufsummiert, sobald sie ungleich Null sind, also eine positive oder negative Hinweiskraft aufweisen. Die Aufsummierung ergibt wieder einen Sicherheitsfaktor im Bereich $[-1, +1]$.

Parallele Aufsummierung von Sicherheitsfaktoren

Sei CF_i der berechnete Sicherheitsfaktor der betrachteten Diagnose und S_i der Sicherheitsfaktor des i-ten Symptoms der betrachteten Verdachtsdiagnose, dann berechnet sich der Sicherheitsfaktor CF der betrachteten Verdachtsdiagnose durch:

$$
\begin{aligned}
CF_0 &= 0 \\
CF_i &= \begin{cases} CF_{i-1} + S_i - CF_{i-1} * S_i & \text{wenn } CF_{i-1}, S_i > 0 \\ CF_{i-1} + S_i + CF_{i-1} * S_i & \text{wenn } CF_{i-1}, S_i < 0 \\ \frac{CF_{i-1}+S_i}{1-\min\{|CF_{i-1}|,|S_i|\}} & \text{sonst} \end{cases}
\end{aligned}
$$

Pitsch weisst in [39, Seite 25f.] darauf hin, dass es einen Sonderfall bei der Berechnung der Sicherheitsfaktoren gibt, in welchem eine Division durch Null erfolgt. Dieser Sonderfall tritt auf, wenn eine Verdachtsdiagnose sowohl Sicherheitsfaktoren mit den Hinweisekräften -1 als auch $+1$ enthält. Mathematisch drückt sich diese Problematik wie folgt aus:

$$
\begin{aligned}
\frac{1+(-1)}{1-\min\{|1|,|(-1)|\}} &= \frac{0}{0} \qquad \text{wenn } CF_{i-1} = 1, S_i = -1 \\
\frac{(-1)+1}{1-\min\{|(-1)|,|1|\}} &= \frac{0}{0} \qquad \text{wenn } CF_{i-1} = -1, S_i = 1
\end{aligned}
$$

In [39, Seite 26] ist weiters beschrieben, dass dieser Sonderfall nur in einem Fall in der aktuellen Wissensbasis von RHEUMexpert auftreten kann. Es sei hier nur vermerkt, dass man mit dem neuen Editor der Wissensbasis beliebige Sicherheitsfaktoren im Bereich $[-1, +1]$ wählen kann, und dieser Fall somit in Zukunft ebenfalls nicht prinzipiell ausgeschlossen werden kann.

Die Sicherheitsfaktoren waren bisher in einer Matrix mit 76x20 Feldern gespeichert. Um aber eine solche Matrix abarbeiten zu können, sind Schleifenkonstrukte in der Programmiersprache notwendig. Da Schleifen aber explizit

unerwünscht sind (der graphische Editor von Kronberger kann diese nicht darstellen), wurde eine andere Art der persistenten Speicherung gewählt.

Kronbergers Editor erzeugt nach dem gleichen Schema wie das Umwandlungsprogramm Funktionsaufrufe, welche die Matrix-Speicherung überflüssig machen. Mit zwei Funktionen werden die für jede Verdachtsdiagnose vorkommenden Sicherheitsfaktoren umgesetzt.

Hierbei wird die Funktion `_cfinner` verwendet, um jeden Sicherheitsfaktor mit einem Symptom zu verknüpfen. Nun wird ein Array mit den Ergebnissen an `_cfouter` übergeben, welche parallel aufsummiert werden.

Eine gekürzte Gegenüberstellung der alten XML- und der neuen Lua-Regeln bietet Listing 3.7.

Listing 3.7: Darstellung der alten XML-Sicherheitsfaktor-Matrix

```
<CF visible="-1">
  <DIAG index="13" />
  <SYM index="1" />
  <VAL value="0.0" />
</CF>
<CF visible="-1">
  <DIAG index="13" />
  <SYM index="2" />
  <VAL value="0.3" />
</CF>
 [...]
<CF visible="-1">
  <DIAG index="13" />
  <SYM index="75" />
  <VAL value="0.9" />
</CF>
<CF visible="-1">
  <DIAG index="13" />
  <SYM index="76" />
  <VAL value="0.5" />
</CF>
```

Listing 3.8: Darstellung der neuen Lua-Sicherheitsfaktor-Abarbeitung

```
function CF13 ()
  return _cfouter({_cfinner(SYM001(),0.0), _cfinner(SYM002 (),0.3),[...],
                   _cfinner (SYM075(),0.9), _cfinner(SYM076(),0.5)})
end
```

Es sei vermerkt, dass `_cfinner` den `Booleschen` Wert des ersten Parameters als Zahl verwendet und in der aktuellen Version diesen mit dem zweiten Parameter mittels der Funktion `_mult` multipliziert. Eine Kombination von `_cfinner` und `_cfouter` erlaubt die Umsetzung beinahe jedes beliebigen Aufsummierungsalgorithmus. In späteren Arbeiten ist es somit möglich andere Algorithmen zur parallelen Aufsummierung zu erstellen, und diese mit geringem Aufwand umzusetzen.

Funktionen die aktuell in den Regeln des dritten Schrittes verwendet werden:

- `_cfinner` Verknüpfung der Sicherheitsfaktoren mit einem Symptom
- `_cfouter` Paralleles Aufsummieren der Werte von `_cfinner`

Eine genauere Beschreibung der Funktionen kann in Kapitel 3.3.1 gefunden werden. Der Quellcode der beiden Funktionen ist im Listing B.1 zu finden.

3.8 Schritt 4: Korrektur der Sicherheitsfaktoren

Im vierten Schritt wird eine Korrektur der Sicherheitsfaktoren der Verdachtsdiagnosen durchgeführt. Für jede Verdachtsdiagnose existiert genau ein Korrekturfaktor. Jeder der für die 20 Verdachtsdiagnosen berechneten Sicherheitsfaktoren wird mit dem für die Verdachtsdiagnose definierten Korrekturfaktor multipliziert, womit als Ergebnis dieses Arbeitsschrittes sämtliche der 20 Sicherheitsfaktoren korrigiert werden [39, Seite 27f.]. Der schon in der früheren Version von RHEUMexpert/Web verwendete Algorithmus wurde in Lua überführt und in seiner Funktionalität nicht verändert.

Es wurden zwei weitere Möglichkeiten zur Korrektur der Sicherheitsfaktoren realisiert. Einerseits gibt es eine Version des vierten Schrittes, welche die Korrekturfaktoren – ähnlich der Sicherheitsfaktorzuordnung im dritten Schritt – persistent gespeichert hat. Es muss aber angemerkt werden, dass

diese Speicherung der errechneten Korrekturfaktoren zwar wesentlich performanter ist, aber beim Hinzufügen von weiteren Symptomen oder Verdachtsdiagnosen neu berechnet werden muss.

Der zweite vorzustellende Algorithmus des vierten Schrittes soll für zukünftige Arbeiten die Flexibilität zum Abschalten der Korrekturfaktoren bieten, um deren Auswirkung und Zweckmäßigkeit genauer untersuchen zu können. Hierbei wird der Korrekturfaktor jedes errechneten Sicherheitsfaktors auf 1 gesetzt um somit die ermittelten Sicherheitsfaktoren der Verdachtsdiagnosen nicht zu verändern.

Diese drei verschiedenen Herangehensweisen haben je eine aufrufbare Lua-Funktion, welche das jeweilige Verfahren umsetzt. Standardmässig wird der originale RHEUMexpert-II-Algorithmus verwendet. Dies kann mit der Übergabe der Programmzeile `step4=funktionsname`, von Java nach Lua, geändert werden.

Die Funktionsnamen sind:

- `step4_original_calculated` Dies ist die Lua-Implementierung des originalen RHEUMexpert-II-Algorithmus.
- `step4_multiply_by_one` Die Korrekturwerte des vierten Schrittes werden mit dieser Funktion ausser Acht gelassen.
- `step4_original_static` Diese Funktion verwendet statisch errechnete Korrekturwerte.

Die vorliegende Arbeit hatte eine Flexibilisierung von RHEUMexpert/Web zur Aufgabe. Die Fragestellung, in welcher Weise Korrekturfaktoren in zukünfigen Versionen des Programmsystems verwendet werden sollen, muss mit einer genügend grossen Menge an Testfällen wissenschaftlich beantwortet werden.

3.9 Schritt 5: Auswahl der Verdachtsdiagnosen

Im Schritt 5 (Auswahl der anzuzeigenden Verdachtsdiagnosen) wurden im Zuge dieser Arbeit einige Algorithmen aus dem Gebiet der digitalen Bildverarbeitung – im genaueren zur Segmentierung von Bildern – auf die Eignung

für RHEUMexpert/Web untersucht. Diese werden des weiteren angeführt, und deren Vor- und Nachteile bzw. der Grund ihrer Nichtverwendung beschrieben.

Um diese Algorithmen zu testen wurde MicroSoft Excel zum rapid Prototyping verwendet, da sich hierin die Funktionsweise der Algorithmen mit allen Zwischenergebnissen leicht veranschaulichen lässt. Nachdem die geeignetsten Algorithmen ausgewählt waren, wurden diese in Lua überführt.

Es wurde im Verlauf dieser Arbeit anhand des Quellcodes erkannt, dass RHEUMexpert beim Anzeigen der Subdiagnosen nur dann aktiv wird, wenn die entsprechende Hauptdiagnose angezeigt wird. Dies steht zwar nicht zwingend im Gegensatz zum Konzept der Sicherheitsfaktoren; dennoch werden Haupt- und Subdiagnosen systemintern – also vom Aufbau der Regeln, bis hin zur Sicherheitsfaktormatrix – in allen Versionen von RHEUMexpert gleichwertig angesehen, und erst im letzten Schritt der Inferenz in deren Bedeutung getrennt. In der Literatur wurde dieses (nachträgliche) Unterteilen von Verdachtsdiagnosen vom Autor dieser Arbeit nicht nochmals gefunden.

Im Zuge der Fertigstellung der neuen Lua-Inferenzmaschine wurde aber die Frage nach dem wissenschaftlichen Hintergrund dieses Vorgehens aufgeworfen, und durch flexible Erweiterungen nun die Möglichkeit zum Vergleich verschiedener Varianten ermöglicht.

3.9.1 Konstanter Schwellwert

Die einfachste Variante ist die manuelle Wahl eines konstanten Schwellwerts für die Verdachtsdiagnosen (z. B. 0,6). Dies ist aber eine unzureichend gute Wahl, da ein fixer Wert für den Schwellwert bei verschiedenen Patienten mit unterschiedlichen Symptomen kritisch ist. Ein unpassender Wert kann schnell zum Ergebnis werden. Führt man das Verfahren mit einem ungeeigneten Wert durch, so erhält man schlechte oder gar keine Ergebnisse – also die Anzeige unwichtiger Verdachtsdiagnosen oder die Nichtanzeige wichtiger Diagnosen [50]. Dies wurde schon in früheren Versionen von RHEUMexpert erkannt, deswegen wird dieses Verfahren nicht umgesetzt.

3.9.2 Mittelwert-Algorithmus

Eine weitere einfache Variante sieht vor, den Mittelwert aller positiver und negativer Sicherheitsfaktoren zur Trennung der anzuzeigenden von den auszuschliessenden Verdachtsdiagnosen zu verwenden. Hierbei ist der Schwellwert der Trennung ident zum Mittelwert. Die Überlegung dahinter ist, dass der Mittelwert sehr anfällig für Ausreisser ist. Es sollen aber in RHEUMexpert genau diese positiven und negativen Ausreisser angezeigt werden. Dies sind Verdachtsdiagnosen welche wesentlich höhere oder geringere Sicherheitsfaktoren aufweisen als die um den Wert Null gruppierten unwichtigen Verdachtsdiagnosen.

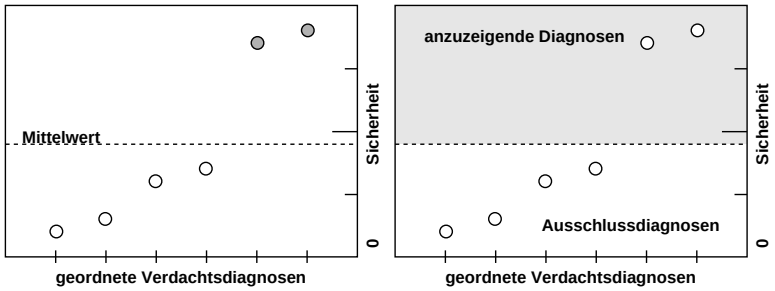

Abbildung 3.3: Arbeitsweise des Mittelwert-Algorithmus

Wie in Abbildung 3.3 zu erkennen ist, liegt der Mittelwert nahe an den auszuschliessenden Diagnosen. Falls mehr Diagnosen hinzukommen, besteht die Möglichkeit den Mittelwert auf drei Arten zu verschieben.

Die erste und weniger rechenintensive Verschiebung des Mittelwertes geschieht durch Addition eines fixen Wertes zum Mittelwert. In „kleinen“ Bereichen (0,1 oder 0,2 bei den Sicherheitsfaktoren) kann dies zu besseren Ergebnissen führen (siehe Abbildung 3.4), dennoch darf nicht vergessen werden, dass sich der Algorithmus nach Hinzufügung einer zu grossen Verschiebung nicht besser verhält, als der in 3.9.1 vorgestellte Algorithmus eines konstanten Schwellwertes.

Bei der zweiten Möglichkeit der Verschiebung des errechneten Mittelwertes werden Stützstellen hinzugefügt. Diese sollen dem Maximalwert im Bereich $[-1, +1]$ entsprechen. Selbstverständlich müssen vor der Anzeige des Ergebnisses diese Stützstellen wieder entfernt werden. Hierbei sollte man aber

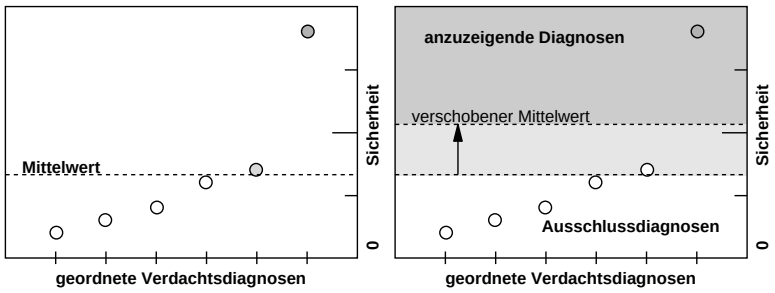

Abbildung 3.4: Verschiebung des Mittelwerts

ebenfalls nicht vergessen, dass der Schwellwert beim Hinzufügen von zu vielen Stützstellen so sehr verschoben wird, dass die Anzeige keines Ergebnisses und damit die Nachteile eines konstanten Schwellwerts zu erwarten sind.

Die dritte Methode zur Verschiebung besteht darin, die Standardabweichung der Verdachtsdiagnosen zum Mittelwert hinzuzufügen. Im Fall von Subdiagnosen (welche geringe Abstände zueinander haben) könnte das Hinzufügen der halben Standardabweichung besser geeignet sein. Es erscheint sinnvoll in weiteren Arbeiten Aufmerksamkeit auf dieses Verfahren zu richten, da es sich zur Auswahl der Verdachtsdiagnosen als geeignet erweisen könnte.

Ohne Verschiebungen wird folgendermassen vorgegangen:

- Schritt 1: Alle Diagnosen werden entsprechend ihrer Hinweis- bzw. Wegweiskraft in positive und negative Gruppen eingeteilt.
- Schritt 2: Die beiden Gruppen werden aufsteigend sortiert.
- Schritt 3: Es wird der Mittelwert aller Sicherheitsfaktoren der Verdachtsdiagnosen errechnet.
- Schritt 4: Alle Werte die grösser als der Mittelwert sind, werden als anzuzeigende Verdachtsdiagnosen gespeichert.
- Schritt 5: Obige Schritte werden für die wegweisenden Werte im negativen Wertebereich wiederholt, wobei die ermittelten anzuzeigenden Werte als Ausschlussdiagnosen gespeichert werden können.

Schritt 5 wird um die gewählte Verschiebung des Mittelwertes erweitert, wenn die Wahl auf eines der drei Verschiebungsverfahren gefallen ist.

3.9.3 RHEUMexpert-II QuasiGradientenAlgorithmus

Das für die Anwendung in RHEUMexpert-II abgeänderte Gradientenverfahren, welches eine Auswahl der 20 Verdachtsdiagnosen trifft die schließlich als Ergebnis ausgegeben werden, wurde im Zuge dieser Arbeit ebenfalls implementiert. Es erstellt einen dynamischen Schwellwert für die berechneten Sicherheitsfaktoren der Verdachtsdiagnosen. Die Berechnung läuft wie folgt ab [39, Seite 40f.]:

- Schritt 1: Alle Hauptdiagnosen werden absteigend nach ihren Sicherheitsfaktoren geordnet.
- Schritt 2: Die Differenzen der nun benachbarten Sicherheitsfaktoren werden berechnet und jeweils der Verdachtsdiagnose mit niedrigerem Sicherheitsfaktor zugeordnet. Die Verdachtsdiagnose mit dem höchsten Sicherheitsfaktor hat demnach keine zugeordnete Differenz, da sie bei absteigender Sortierung der Verdachtsdiagnosen keinen Vorgänger hat.
- Schritt 3: Diese Differenzen werden jeweils durch den ganzzahligen Abstand ihrer zugeordneten Verdachtsdiagnose zur Verdachtsdiagnose mit dem höchsten Sicherheitsfaktor dividiert. Das Ergebnis wird als gewichtete Differenz bezeichnet und der Verdachtsdiagnose zugeordnet, welcher auch die Differenz aus Schritt 2 zugeordnet ist. Die erste Verdachtsdiagnose hat keine zugeordnete Differenz. Die der zweiten Verdachtsdiagnose zugeordnete Differenz wird durch die Zahl 1 dividiert, bleibt also unverändert. Die der dritten Verdachtsdiagnose zugeordnete Differenz wird durch die Zahl 2 dividiert, usw..

  Als Anmerkung möchte der Autor festhalten, dass ihn dies an ein Verfahren aus der Statistik bei multiplen Tests, welches als Bonferoni-Holm-Korrektur bekannt ist, erinnert. Dies sei aber nur vermerkt, um interessierten Lesern die Möglichkeit zu bieten, tiefer in das Themengebiet der statistischen Tests einzudringen und selbst einen Vergleich mit dem hier angewendeten Verfahren in [25] zu ziehen.
- Schritt 4: Zwischen den beiden Verdachtsdiagnosen, in denen die in Schritt 3 berechnete gewichtete Differenz am größten ist, wird die Darstellungsgrenze (Schwellwert) gesetzt.

- Schritt 5: Für jede Hauptdiagnose werden nun auch die Schritte 1 bis 4 für die Unterdiagnosen durchgeführt. Es werden also für die Hauptdiagnosen und für jede Unterdiagnosegruppe jeweils ein Schwellwert berechnet. Unter einer Unterdiagnosegruppe wird hier die Menge aller Unterdiagnosen verstanden, die derselben Hauptdiagnose zugeordnet sind.
- Schritt 6: Zur Anzeige gelangen schließlich, neben den Ausschlussdiagnosen, alle Hauptdiagnosen, deren Sicherheitsfaktor über dem im vierten Schritt ermittelten Schwellwert liegen. Wird eine Hauptdiagnose angezeigt, so werden auch ihre zugehörigen Unterdiagnosen angezeigt, deren Sicherheitsfaktoren über dem ermittelten Schwellwert für die entsprechende Unterdiagnosegruppe liegen (siehe Schritt 5).

Wie Pitsch in [39, Seite 29] bemerkt, weist der Algorithmus auch Schwächen auf. Diese sind:

- Sofern mehrere Verdachtsdiagnosen mit positivem Sicherheitsfaktor berechnet wurden, wird auf Grund von Schritt 4 immer jedenfalls eine dieser Verdachtsdiagnosen nicht angezeigt, unabhängig vom Wert des Sicherheitsfaktors.
- Existiert überhaupt nur eine einzige Verdachtsdiagnose mit positivem Sicherheitsfaktor, so kann in Schritt 2 keine Differenz berechnet werden, womit in Folge in Schritt 3 auch keine gewichtete Differenz ermittelbar ist.
- Schließlich ist in Schritt 4 kein Schwellwert ermittelbar und es existiert keine Anweisung, wie mit dieser Situation umzugehen ist.

Ein weiteres, eher theoretisches Szenario tritt ein, wenn sämtliche gewichteten Differenzen aus Schritt 3 ident sind. Für diesen Fall gibt es in Schritt 4 ebenfalls keine vorgegebene Handlungsanweisung; das Ergebnis ist damit undefiniert. Durch die Multiplikation der Abstände (Gradienten) der einzelnen Verdachtsdiagnosen mit deren Position kann dieses Verfahren nur als Quasi-Gradientenverfahren bezeichnet werden.

3.9.4 Gradientenverfahren

Ein striktes Gradientenverfahren, welches die grösste Steigung der geordneten Verdachtsdiagnosen als Schwellwert verwendet, wurde ebenfalls in Lua umgesetzt. Auch in diesem Algorithmus werden Stützstellen im Bereich von 0 und 1 verwendet, um mit Fällen umgehen zu können, in denen keine Verdachtsdiagnosen sinnvoll anzuzeigen sind.

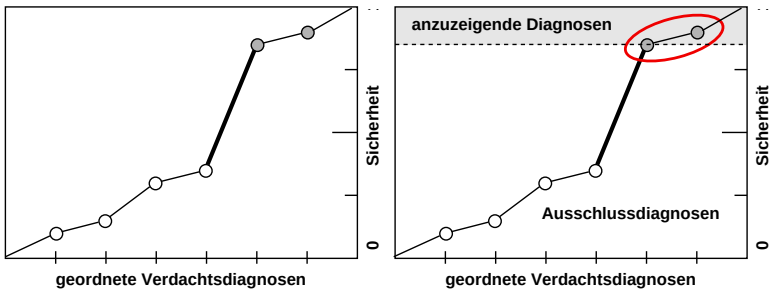

Abbildung 3.5: Arbeitsweise des Gradienten-Algorithmus

Wie man in Abbildung 3.6 sehen kann schliesst die umgesetzte Version im Fall zweier gleicher Gradienten den mit dem geringeren Sicherheitsfaktor aus. Dieses Verhalten kann durch die Änderung des Vergleichsoperators im Algorithmus leicht angepasst werden, so dass im Fall zweier gleicher Gradienten auch der mit dem geringeren Sicherheitsfaktor angezeigt wird.

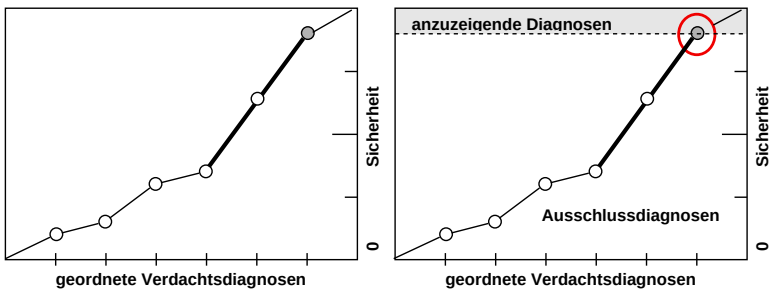

Abbildung 3.6: Gradienten-Algorithmus bei gleichen Gradienten

In Abbildung 3.7 sieht man einerseits, dass im Fall eines grösseren Gradienten zu den Stützstellen kein bzw. jedes Ergebnis ausgegeben wird. Man er-

kennt an diesem letzen Beispiel, dass der Algorithmus durchaus seine Schwächen hat. Dies ist wohl eine Begründung dafür, dass in RHEUMexpert-II ein verändertes Verfahren umgesetzt wurde.

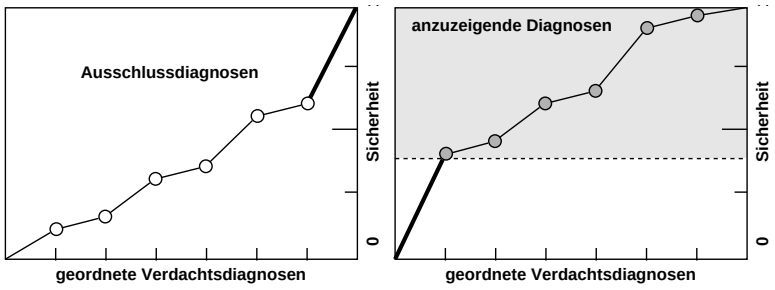

Abbildung 3.7: Extremfälle des Gradienten-Algorithmus

Die Berechnung wird wie folgt durchgeführt:

- Schritt 1: Alle Diagnosen werden entsprechend ihrer Hinweis- bzw. Wegweiskraft in positive und negative Gruppen eingeteilt.
- Schritt 2: Die beiden Gruppen werden aufsteigend sortiert.
- Schritt 3: Es werden Stützstellen bei 0 und 1 eingefügt, um im Fall keines anzuzeigenden Ergebnisses korrekt zu arbeiten.
- Schritt 4: Mit einer Schleife wird über jeden der Werte (absteigend) iteriert und sowohl die Indexposition, als auch die Differenz zum Nächsten gespeichert, wenn diese grösser als die vorherige Differenz ist.
- Schritt 5: Alle indizierten Werte, die grösser als der gespeicherte Index sind, werden als anzuzeigende Verdachtsdiagnosen gespeichert.
- Schritt 6: Obige Schritte werden für die ermittelten wegweisenden Sicherheitsfaktoren im negativen Bereich wiederholt, wobei die ermittelten anzuzeigenden Werte als Ausschlussdiagnosen gespeichert werden können.

Wie zu erkennen ist, weist auch dieser Algorithmus konzeptionelle Schwächen auf. Dennoch sollte in weiteren Arbeiten Tests – mit einer genügend grossen Anzahl von Goldstandard-Fällen – gemacht werden, um Aufschluss über dessen Verwendbarkeit zu erhalten.

3.9.5 Otsu-Algorithmus

Eine in der Bildverarbeitung aufwendigere Methode zur Berechnung des passenden Schwellwertes für ein Bild ist das Otsu-Verfahren [50]. Diese Methode setzt voraus, dass im Histogramm eine bimodale Verteilung der Grauwerte vorhanden ist [41], das heisst, es gibt im Histogramm zwei Gipfel. Der eine enthält die Pixel des Hintergrunds und der andere die Pixel der Objekte. Diese Voraussetzung ist im Fall von RHEUMexpert/Web gegeben, da Verdachtsdiagnosen entweder angezeigt, oder nicht ausgegeben werden sollen. Die Berechnung geschieht dann wie folgt. Ein Maß für die Homogenität einer Region, also auch eines Peaks, ist die Varianz – eine homogene Region hat eine geringe Varianz. Man berechnet den Schwellwert, indem die Varianz innerhalb der beiden Klassen minimiert wird, die durch den Schwellwert (engl. Threshold) getrennt werden.

Genau hier liegt aber im Fall von RHEUMexpert/Web das Problem, da ungenügend viele Verdachtsdiagnosen (sieben Hauptdiagnosen, dreizehn Subdiagnosen) zur Verfügung stehen, aus welchen keinesfalls ein Histogramm berechnet werden kann. Dieser Algorithmus war dennoch der Erfolgversprechendste, wenn auch in einer vereinfachten Form, welche nicht mit Klassen im Histogramm, sondern direkt mit den Sicherheitsfaktoren der Verdachtsdiagnosen auskommt und im nächsten Abschnitt genauer beschrieben ist.

3.9.6 Vereinfachter Otsu-Algorithmus

Das für die Anzeigeentscheidung der Verdachtsdiagnosen erstellte Verfahren verwendet Teile des Otsu-Algorithmus, aber nur soweit diese nicht mit den klassifizierten Bereichen des Histogramms, sondern direkt mit den sieben Haupt- und dreizehn Subdiagnosen arbeiten. Hierbei werden die Varianzen der einzelnen Gruppen eines dynamisch wandernden Schwellwertes verglichen und der Schwellwert, der die beiderseits geringsten Varianzen ergibt, wird verwendet.

In Abbildung 3.8 sind vier der sieben Fälle zu sehen. Es ist klar, dass die Varianz der einen Gruppe (in den ersten beiden Graphiken) Null ist. Die Varianz der anderen Gruppe ist dafür umso grösser. Da sämtliche Gruppen-

grössen iteriert werden, findet der Algorithmus in der letzten Graphik die Lösung des Problems – die geringsten Varianzen der beiden Gruppen.

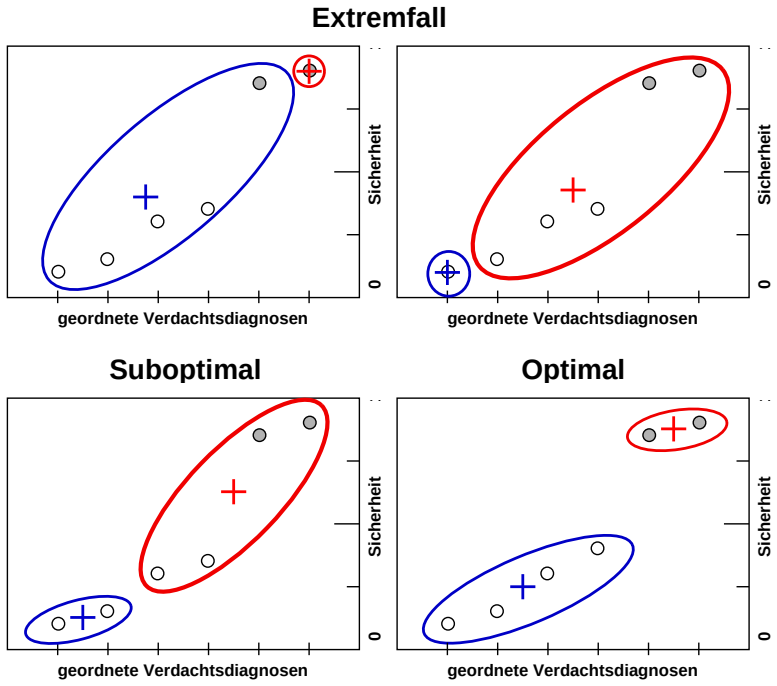

Abbildung 3.8: Vorgehensweise des vereinfachten Otsu-Algorithmus

Die Berechnung wird wie folgt durchgeführt:

- Schritt 1: Alle Diagnosen werden entsprechend ihrer Hinweis- bzw. Wegweiskraft in positive und negative Gruppen eingeteilt.
- Schritt 2: Die beiden Gruppen werden aufsteigend sortiert.
- Schritt 3: Es werden Stützstellen bei dem minimalen und maximalen Sicherheitsfaktor der Verdachtsdiagnosen eingefügt, um die Gruppen besser voneinander zu trennen.
- Schritt 4: Es wird der Mittelwert der anzuzeigenden und vernachlässigbaren Gruppen gebildet, welche durch den Schwellwert X getrennt werden. X wird für jeden der geordneten Werte gesetzt/iteriert.
- Schritt 5: Es werden die Varianzen für beide Gruppen (anzuzeigende vs. vernachlässigbare Ergebnisse) errechnet und deren absolute Differenz gespeichert.

Dieses Verfahren wird verwendet um Aussagen über die Homogenität der Gruppen bilden zu können.

- Schritt 6: Der Schwellwert X wird erhöht und eine weitere Iteration der Schritte 3–5 findet statt.
- Schritt 7: Nachdem sämtliche Varianzen aller theoretisch möglichen Gruppen gebildet sind, wird die kleinste gewählt, wodurch die homogensten Gruppen gefunden sind und damit auch der optimale Schwellwert ermittelt ist.
- Schritt 8: Alle Werte, die grösser als der Schwellwert sind, werden als anzuzeigende Verdachtsdiagnosen gespeichert.
- Schritt 9: Selbige Schritte werden für die wegweisenden Werte im negativen Wertebereich wiederholt, wobei die ermittelten anzuzeigenden Werte als Ausschlussdiagnosen gespeichert werden können.

Es gibt für dieses Verfahren Möglichkeiten die Ergebnisse des Algorithmus – also wieviele Verdachtsdiagnosen/Ausschlussdiagnosen angezeigt werden – zu verändern. Durch Hinzufügen anderer Stützstellen im hohen Wertebereich (0,5 bis 1), oder durch ein zusätzliches Hinzufügen des höchsten Sicherheitsfaktors als Stützstelle, wird die Gruppe von Diagnosen im Bereich der Stützstelle homogener. Hierdurch würde der Schwellwert für die anzuzeigende Gruppe weiter nach oben verschoben werden. Selbstverständlich müssen diese Stützstellen vor der Anzeige des Ergebnisses wieder entfernt werden. Hierbei sollte man aber nicht vergessen, dass der Schwellwert beim Hinzufügen von zu vielen Stützstellen zu sehr verfälscht wird, dass entweder keine, oder nur die Verdachtsdiagnose mit dem höchsten Sicherheitsfaktor, angezeigt wird.

3.9.7 Zusammenfassung

Wir haben im aktuellen Kapitel eine Vielzahl verschiedener (unterschiedlich gut geeigneter) Algorithmen untersucht. Ähnlich wie im vierten Schritt kann auch im fünften von der evaluierenden Person ausgewählt werden, welcher Algorithmus verwendet werden soll.

Die Algorithmen zur Auswahl der Verdachtsdiagnosen haben jeweils eine Funktion zu ihrer Umsetzung. Es wird standardmässig der originale RHEUMexpert-II Algorithmus verwendet. Dies kann aber mit der Übergabe der Programmzeile `step5=funktionsname`, von Java nach Lua (oder mit dem Einfügen dieser Zeile in das File `inferenz.lua`), geändert werden.

Die Namen der bereitgestellten Funktionen sind:

- `step5_original` Der RHEUMexpert-II Algorithmus aus 3.9.3
- `step5_meanvalue` Der neue Mittelwert-Algorithmus aus 3.9.2
- `step5_gradient` Das Gradientenverfahren aus 3.9.4
- `step5_simple_otsu` Der vereinfachte Otsu-Algorithmus aus 3.9.6

Im Zuge der Fertigstellung dieser Arbeit wurden die Flexibilität des Tests verschiedener Algorithmen des fünften Schrittes (der Inferenz) realisiert. In weiteren Arbeiten kann durch diese Flexibilisierung möglicherweise eine Entscheidung für den idealsten Algorithmus getroffen werden.

3.10 Einbinden der Lua-Regeln in Java

In den letzten Abschnitten haben wir die neuartige Inferenz und deren Vorteile behandelt. Die entscheidende Frage ist aber, wie sich die neuen Regeln in den bestehenden Java-Server einbinden lassen [5].

Hier kommt uns das Softwarepaket LuaJava – welches schon in Kapitel 2.8.2 näher beschrieben wurde – zu Hilfe. Mit LuaJava besitzt man ein Framework um sämtliche Inferenzfunktionen in Lua von Java auszuführen bzw. aufzurufen. Weiters kann man die XML-Rückgaben der Lua-Inferenz in Java empfangen, wodurch die Inferenzfunktion durch Ihren Aufruf schon das Ergebnis liefert.

Durch die neuartige Inferenz sind die alten XML-Inferenz-Klassen obsolet geworden. Hierdurch verwendet der neue Java-Server weniger Resourcen. Um LuaJava in den Java-Quellcode einzufügen bedarf es einiger kleiner Änderungen:

- Einbinden der LuaJava JAR- und DLL-Datei über den CLASS-Path (z. B. der Eclipse-Workspace-Umgebung).
- Hinzufügen der Lua-Inferenz-Aufrufe in die Konsultationsklasse.

Den zweiten Punkt der Änderungen wollen wir näher betrachten, da dieser die eigentliche Arbeit am Java-Server bezeichnet. Wie schon erwähnt erhält man als Rückgabe der Inferenz ein XML-Ergebnis in der Form von:

Listing 3.9: XML-Ergebnis der Inferenz

```
<Result>
  <DIAG index="1" show="1" certainty="0.8433" />
  <DIAG index="2" show="0" certainty="0.1" />
  <DIAG index="3" show="0" certainty="0.001" />
   [...]
  <DIAG index="20" show="1" certainty="-1" />
</Result>
```

Dieses Ergebnis muss im Java-Server weiterverarbeitet und an den Client gesendet werden. In den Listings 3.10 und 3.11 sieht man eine Gegenüberstellung des alten und des neuen Java-Konsultations-Quellcodes. Man erkennt darin die Änderungen am Programmcode im Detail. Es sei vermerkt, dass ungeänderte Programmteile nicht angezeigt werden und durch `/* [...] */` ersetzt sind.

Listing 3.10: Alter Java-Server-Quellcode

```
\begin{ lstlisting }[fontadjust]
package rh2j.sys_consultation.server;
/* [...] */
import org.keplerproject.luajava.LuaState;
import org.keplerproject.luajava.LuaStateFactory;
/* [...] */
public class ConsultationSystemServer extends AbstractForSubsystemLogics{
  private static Logger logger = Logger.getLogger(ConsultationSystemServer.class);
  public ConsultationSystemServer(){ /* [...] */ }
  public void takeOverProcessControl(){
    /* [...] */
    try{
      inference = new Inference();
      int i = 0;
      while(true){ // do until connection gets closed by the client
        Date timeRequest1 = new Date();
        request = this.receiveDocument(); // IOException if client kills connection
        // request blocks when the conn to the client gets lost the thread concludes

        logger.info("handling request # "+(++i));

        inference.infer(convertInputFromJdomToHashtable(request.getRootElement()));
        // prepare results for sending
        Document docSde = convertOutputFromHashtableToJdom_sde (inference.getSDE());
        Document docIsde = convertOutputFromHashtableToJdom_isde(inference.getISDE());
        Document docSym = convertOutputFromHashtableToJdom_sym (inference.getSYM());
        Document docDiag = convertOutputFromHashtableToJdom_diag(inference.getDIAG());
        // send results
        this.sendDocument(docSde);
        this.sendDocument(docIsde);
        this.sendDocument(docSym);
        this.sendDocument(docDiag);

        Date timeRequest2 = new Date();
        logger.info("request # "+i+" process time: "+Tools.timeDifferenceFormat(timeRequest1, timeRequest2)+")");
      }
    }
    catch(EOFException e){ logger.info ("connection closed by other host"); }
    catch(Exception e){    logger.error("error handling document.", e);      }
    Date time2 = new Date();
    logger.info("CONSULTATION END (process time: "+Tools.timeDifferenceFormat(time1, time2)+")");
  }
  private Hashtable convertInputFromJdomToHashtable(Element root)throws Exception{ /* [...] */ }
  private Document convertOutputFromHashtableToJdom_sde(Hashtable local){ /* [...] */ }
  private Document convertOutputFromHashtableToJdom_isde(Hashtable local){ /* [...] */ }
  private Document convertOutputFromHashtableToJdom_sym(Hashtable local){ /* [...] */ }
  private Document convertOutputFromHashtableToJdom_diag(Hashtable local){ /* [...] */ }
}
```

Listing 3.11: Neuer Java-Server-Quellcode

```
package rh2j.sys_consultation.server;
/* [...] */
import org.keplerproject.Luajava.LuaState;
import org.keplerproject.Luajava.LuaStateFactory;
/* [...] */
public class ConsultationSystemServer extends AbstractForSubsystemLogics{
  private static Logger logger = Logger.getLogger(ConsultationSystemServer.class);
  public ConsultationSystemServer(){ /* [...] */ }
  public void takeOverProcessControl(){
    /* [...] */
    try{
      inference = new Inference();
      int i = 0;
      while(true){ // do until connection gets closed by the client
        Date timeRequest1 = new Date();
        request = this.receiveDocument(); // IOException if client kills connection
        logger.info("handling request # "+(++i));
        LuaState L = LuaStateFactory.newLuaState();
        L.openBasicLibraries();
        List xmlList = request.getRootElement().getChildren();
        Element xmlElement;
        Iterator iterator = xmlList.iterator();
        while (iterator.hasNext()){ // iterate xml-request and eval vars to Lua-vars
          xmlElement = (Element) iterator.next();
          Integer index = new Integer(xmlElement.getAttribute("index").getIntValue());
          String indexx = "";
          for (n=3; n>index.toString().length(); n--) indexx+= "0";
          indexx += index.toString();
          // Lua doesn't work with the -1 logic for a not set values (needed in one or two cases)
          // but there is a workaround to use some small value less then 0 to do so
          if (Float.compare(xmlElement.getAttribute("value").getFloatValue(), 0) != -1){
            L.doString("SDE" + indexx + "=" + xmlElement.getAttribute("value").getFloatValue());
          }else{ L.doString("SDE" + indexx + "= -0.00000001"); }
        }
        L.pushString("logger"); L.pushObjectValue(logger); // Lua loggingcapacities
        L.setGlobal("logger");

        int err=L.doFile("inferenz.Lua");                    // run the inference-file
        System.out.print(L.getLuaObject("ret"));             // returnvalue
        L.close();                                           // close Lua-interpreter
        Date timeRequest2 = new Date();
        logger.info("request # "+i+" process time: "+Tools.timeDifferenceFormat(timeRequest1, timeRequest2)+")");
      }
    }
    catch(EOFException e){ logger.info ("connection closed by other host"); }
    catch(Exception e){     logger.error("error handling document.", e);      }
    Date time2 = new Date();
    logger.info("CONSULTATION END (process time: "+Tools.timeDifferenceFormat(time1, time2)+")");
  }
  private Hashtable convertInputFromJdomToHashtable(Element root)throws Exception{ /* [...] */ }
  private Document convertOutputFromHashtableToJdom_sde(Hashtable local){ /* [...] */ }
  private Document convertOutputFromHashtableToJdom_isde(Hashtable local){ /* [...] */ }
  private Document convertOutputFromHashtableToJdom_sym(Hashtable local){ /* [...] */ }
  private Document convertOutputFromHashtableToJdom_diag(Hashtable local){ /* [...] */ }
}
```

Man kann im obigen Listing 3.11 erkennen, dass das Einfügen der Eingabedaten in die LuaJava-Umgebung etwas holprig aussieht. Es gibt zwar die Möglichkeit direkt Variablen von Java in die Lua-Umgebung einzufügen und mit einem Wert zu versehen (was z. B. beim `logger` geschieht). Ab einer Anzahl von ungefähr 100 Variablen (welche in einer Schleife von Java nach Lua gepusht werden) kommt es zu einer Schutzverletzung, da LuaJava scheinbar die Anzahl der Variablen nicht in solch kurzer Zeit verarbeiten kann.

Abhilfe wird erzielt, wenn die Variablen nicht definiert nach Lua gesendet, sondern ähnlich wie beim Java-Befehl `eval` direkt in Lua ausgeführt werden. Da die Rückgabe der `doString`-Funktion abgewartet wird, kommt es nicht zu der Überforderung von Lua. Hierdurch werden bei jeder Berechnung die Eingabevariablen in Lua, für die weitere Verarbeitung, eingefügt.

Nach dem Aufruf von `inferenz.lua` kann das XML-Ergebnis aus der Lua-Variable `ret` ausgelesen, weiterverarbeitet und an den Client gesendet werden.

3.11 Alte versus neue Wissensverarbeitung

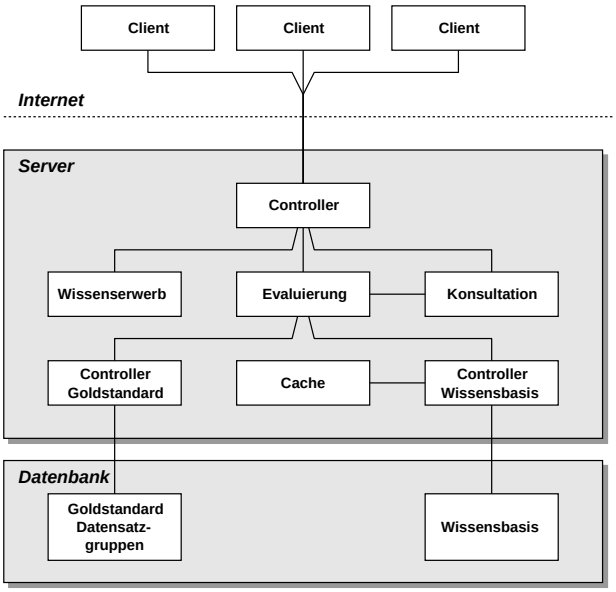

Abbildung 3.9: Aufbau und Abarbeitung des alten RHEUMexpert/Web

Der letzte Teil des Kapitels zeigt eine Gegenüberstellung der alten und neuen Subsysteme von RHEUMexpert. Abbildung 3.9 beschreibt die Herangehensweise von Pitsch. Man sieht darin, die Verwendung des Wissens-Caches und die angedachte Verwendung eines Controllers der Goldstandard-Datensätze.

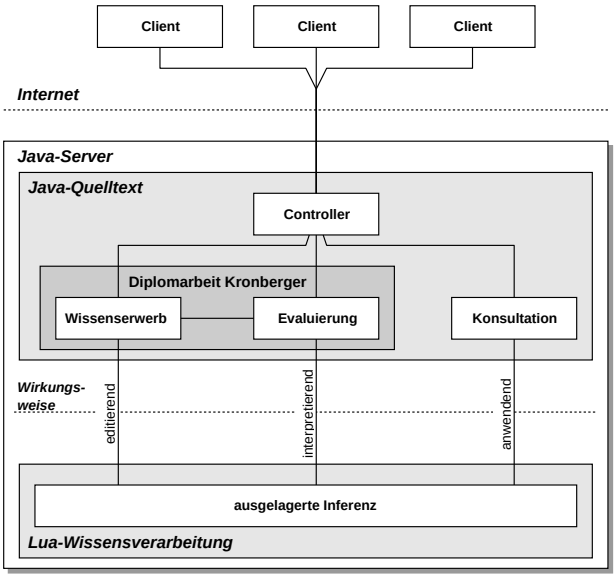

Abbildung 3.10: Aufbau und Abarbeitung des neuen RHEUMexpert/Web

Im Gegensatz dazu sieht man in Abbildung 3.10 die tatsächlich entstandene Abarbeitung der Subsysteme in RHEUMexpert. Man erkennt hierbei auch, das zum Datum der Drucklegung dieser Arbeit keine Anbindung an eine Datenbank – wie in Pitschs Arbeit [39, Seite 57] beschrieben wird – existiert.

3.12 Conclusio

In diesem Kapitel wurde beschrieben, wie die neuartige Inferenz in Lua erstellt, abgearbeitet und an den Java-Server zurückgegeben wird. Man sieht, dass die Namensgebung der Regeln gleich jener aus vorangegangenen Arbeiten ist, dass man die einzelnen Regelfunktionen aber beliebig benennen kann, um die Lesbarkeit zu erhöhen. Es stellt weiters eine Auflistung sämtlicher zur Verwendung stehender Funktionen der Wissensverarbeitung dar und kann als Hilfe für weitere Arbeiten genauso von Nutzen sein wie zum Verständnis der Interaktion zwischen Java und Lua.

4 Resultate

Das Ziel der vorliegenden Arbeit war es, die komplette Inferenz von RHEUMexpert/Web aus der veralteten XML/Programmcode-Form in eine neue Form zu überführen, welche gemeinsam mit Kronbergers Programm [38] Möglichkeiten zum graphischen Editieren der Wissensbasis bietet. Ein weiteres Ziel war es, dem evaluierenden Experten die Möglichkeit zu geben, die Algorithmen zur Korrektur der Sicherheitsfaktoren, und deren Auswahl zur Laufzeit abzuändern.

RHEUMexpert/Web ist ein medizinisches Expertensystem zur Erstellung von Verdachtsdiagnosen in der Rheumatologie. Die früheren Versionen von RHEUMexpert hatten keine Möglichkeit die Wissensbasis anzupassen, da diese direkt im Programmcode realisiert war. In Folge dessen wurde im Zuge der Überarbeitung der Grammatik für RHEUMexpert/Web eine Auftrennung von Programmcode und Wissensverarbeitung vollzogen.

Diese neue Inferenz baut auf der Sprache Lua auf, einer äusserst flexiblen Skriptsprache, die in einer kompakten virtuellen Maschine interpretiert und abgearbeitet wird. Lua kann mit dem Java-Paket LuaJava in den bestehenden RHEUMexpert/Web-Server eingebunden werden. Die eigentliche Wissenssprache wurde in ihrer EBNF definiert und deren Umfang beschrieben.

Die neue Inferenz baut auf eigens erzeugten Operatorfunktionen auf, um leichter für zukünftige Vorhaben editierbar zu sein.

Die Flexibilisierung der Inferenz von RHEUMexpert/Web wurde, wie in den letzten Kapiteln zu lesen war, erreicht. Die Suche nach dem idealsten Algorithmus zur Anzeige der Verdachtsdiagnosen wird durch diese Arbeit massgeblich vereinfacht. Der Test von drei neuen Algorithmen des fünften Schritts der Inferenz, kann durch Übergabe einer einzigen Programmzeile von Java nach Lua, noch zur Laufzeit des Java-Server, realisiert werden, wodurch eine Gegenüberstellung dieser Algorithmen erstmals möglich wird.

Mithilfe des erzeugten Programmcodes ist es möglich, einen graphischen Editor, der Teil einer verwandten Magisterarbeit war, im praktischen Einsatz zu testen, und die Qualität neuer Regeln, oder die Änderung der Sicherheitsfaktoren, noch vor deren Einsatz in RHEUMexpert/Web zu prüfen.

Ob sich der erhoffte Performance-Gewinn des RHEUMexpert/Web-Server wirklich einstellt, kann zum aktuellen Zeitpunkt noch nicht festgestellt werden. Dies wird erst der praktische Einsatz des Systems zeigen können.

Die Anzeige der neuen Lua-Regeln in Kronbergers graphischem Editor hat sich als etwas mühsamer herausgestellt, als noch am Ende seiner Magisterarbeit gehofft wurde. Dies wird damit begründet, dass der Editor nur schwer mit zu grossen Regel-Bäumen umgehen kann, und deren Interpretation und Editierung einige Zeit in Anspruch nimmt.

Zum gegenwärtigen Zeitpunkt sind keine ergebnistechnischen Unterschiede zwischen der Inferenz von Pitsch [39] und der neuen Lua-Inferenz bekannt. Ob alle Ergebnisse der alten und der neuen Wissensmaschine ident sind, muss im praktischen Versuch gezeigt werden.

5 Ausblick

Zum Abschluss wird kurz angeführt, welche Möglichkeiten dem Forschungsteam um RHEUMexpert durch die vorliegende Arbeit in der weiteren Zukunft offen stehen.

5.1 Begutachtung der Korrekturfaktoren

Für weitere wissenschaftliche Untersuchungen ist es durch die vorliegende Arbeit möglich geworden, die Angemessenheit der Korrekturfaktoren zu untersuchen. Durch die Einbindung der Lua-Regeln und mit der Übergabe einer einzigen Variable ist es möglich die Berechnung der Korrekturfaktoren zur Laufzeit abzuändern, oder diese für weitere Versuche einfach auszuschalten.

5.2 Veränderbarkeit des Anzeigealgorithmus

Die vorliegende Arbeit bietet die Möglichkeit, mit der Übergabe einer weiteren Variable, den Algorithmus zur Auswahl der anzuzeigenden Verdachtsdiagnosen auf einen der drei neu erstellten Algorithmen abzuändern. Dies kann vom evaluierenden Experten in Zukunft leicht in einem graphischen Benutzerinterface getan werden, wodurch alle – in der vorliegenden Arbeit – erzeugten Algorithmen auf deren Eignung getestet werden können.

5.3 Lesbarere Regeln

In Kapitel 3.6 sieht man gegen Ende, dass es möglich ist, sämtliche Symptome, Symptomklassen, Diagnosen u. dgl. mit sprechenderen Namen als z. B. `SYM020()` zu benennen. Durch die vorliegende Arbeit ist es möglich alle variablen Bestandteile der Regelfunktionen mit beliebigen Namen zu versehen. Kronbergers Editor unterstützt diese Konvention, wodurch Lua-Regeln möglich sind, die auch von Nichtinformatikern ohne einen speziellen Editor gelesen und verstanden werden können.

5.4 Fuzzy-Logik

Mit den – eigens für die Regeln geschriebenen – Operatorfunktionen ist es möglich, in weiteren Arbeiten RHEUMexpert/Web relativ leicht soweit abzuändern, dass die Verarbeitung der Regeln auf Fuzzy-Logik basiert. Hierfür müssen im besten Fall nur die Operatorfunktionen und das Aufsummieren der Sicherheitsfaktoren, nicht aber die Regeln selbst, abgeändert werden.

5.5 Automatisches Lernen

Mittels eines neuronalen Netzwerkes könnte es möglich sein, in weiteren Arbeiten kleine Änderungen an den Sicherheitsfaktoren selbständig vom Programm durchführen zu lassen. Hierbei könnte ein mehrlagiges Perzeptron (MLP) (engl. Multi Layer Perceptron) mit den Sicherheitsfaktoren aus dem dritten Schritt der Inferenz angelernt werden, und mit Hilfe der Goldstandard-Datensätze – aus Pitschs Doktorarbeit – Änderungen an den gelernten Sicherheitsfaktoren vornehmen. Diese könnten dann ohne Abbruch des Servers verändert werden, um eine neue Iteration des MLPs zu starten. Nach einer genügend grossen Anzahl an Lernschritten könnten die Sicherheitsfaktoren soweit abgestimmt werden, dass sie mit den Goldstandard-Datensätzen zu besseren Diagnosen kommen.

Hiermit wäre RHEUMexpert wohl eines der ersten (wenn nicht das erste) Expertensysteme, welches nicht nur halbautomatisch (wie schon Systeme in der Vergangenheit), sondern vollautomatisch seine eigene Arbeitsweise und die verwendeten Sicherheitsfaktoren verbessert.

Appendix A: XML-Lua-Transformationscode

Listing A.1: PHP-Quellcode zur Umwandlung der XML-Regeln

```
1  <?php
2  define ("_BEGIN",0);          define ("_END",1);
3  define ("GREATER_THAN","0"); define ("GREATER_EQUAL","1");
4  define ("LESS_THAN","0");  define ("LESS_EQUAL","1");
5
6  // lua rules von vorne und hinten (klammer zu usw). rule_e(nd) wird am ende umgedreht, und alles implodet
7  global $rule_b; $rule_b = array();
8  global $rule_e; $rule_e = array();
9  global $ri;
10 global $lookup;
11 $lookup["-"]="_sub";
12 $lookup["+"]="_sum";
13 $lookup[">"]=GREATER_THAN;
14 $lookup[">="]=GREATER_EQUAL;
15 $lookup["<"]=LESS_THAN;
16 $lookup["<="]=LESS_EQUAL;
17
18 class ObjectFromXML{
19   var $parser;
20   var $iter = 0;
21   var $path = array();
22   var $xml = array();
23   var $indizes = array();
24   var $_parent;
25
26   function ObjectFromXML($XML){
27     $this->parser = xml_parser_create();
28
29     xml_set_object($this->parser, &$this);
30     xml_parser_set_option($this->parser, XML_OPTION_SKIP_WHITE, 1);
31     xml_parser_set_option($this->parser, XML_OPTION_CASE_FOLDING, 0);
32     xml_set_element_handler($this->parser, "hanleTagStart", "hanleTagEnd");
33     xml_set_character_data_handler($this->parser, "hanleTagCData");
34     xml_parse($this->parser, $XML);
35     xml_parser_free($this->parser);
36
37     $this->xml = $this->xml['_children'][0];
38
39     $this->_where_index=1;
40     $this->_where_parent = &$this->xml["_children"];
41     $this->_where = &$this->_where_parent[$this->_where_index];
42     $this->_where_first=true;
43   }
44
45   function &getChild($where=""){ return $this->_where; }
46
47   function &getNextChild($where=""){
48     $next=false;
49     foreach ($this->_where_parent as $index => $value) {
50       if ($next==true) {
51         $found=true;
```

```
52          break;
53        }
54        if ($index==$this->_where_index && !$this->_where_first){
55          $next=true;
56        }elseif($index==$this->_where_index && $this->_where_first){
57          $this->_where_first=false;
58          $found=true;
59          break;
60        }
61      }
62      if ($found){
63        $this->_where_index=$index;
64        $this->_where = &$this->_where_parent[$this->_where_index];
65        return $value;
66      }else{
67        return false;
68      }
69    }
70    function getEvalPath(){ return '$this->xml[' . "'" . implode("'][", $this->path) . "'" . ']';}
71
72    function hanleTagStart($parser, $tag, $attributes){
73      array_push($this->path, '_children');
74      array_push($this->path, ($this->iter++));
75
76      $e = $this->getEvalPath();
77      eval ("\$this->indizes['".($this->iter-1)."'] = &".$e.";");
78      eval ($e . "['_name'] = \$tag;");
79      if ($attributes !== array()){
80        eval ($e . "['_attributes'] = \$attributes;");
81      }
82    }
83
84    function hanleTagCData($parser, $cdata){}
85
86    function hanleTagEnd($parser, $tag){
87      $last=0;
88      $count=0;
89      // link auf vater und das erste kind setzen
90      if (sizeof($this->path)>2){
91        $this->indizes[$this->path[sizeof($this->path)-1]]["_parent"] =& $this->indizes[$this->path[sizeof(
            $this->path)-3]];
92        if (@!$this->indizes[$this->path[sizeof($this->path)-1]]["_parent"]["_firstChild"]){
93          $this->indizes[$this->path[sizeof($this->path)-1]]["_parent"]["_firstChild"] =& $this->indizes[$this
              ->path[sizeof($this->path)-1]];
94        }
95        $this->indizes[$this->path[sizeof($this->path)-1]]["_parent"]["_lastChild"] =& $this->indizes[$this->
            path[sizeof($this->path)-1]];
96      }
97      // herausfinden wer der letzte bruder ist
98      if (@$this->indizes[$this->path[sizeof($this->path)-1]]["_parent"]){
99        foreach ($this->indizes[$this->path[sizeof($this->path)-1]]["_parent"]["_children"] as $index => $value)
            {
100         $count++;
101         if ($index!=$this->path[sizeof($this->path)-1]) $last=$index;
102       }
103     }
```

```
104       // verlinken von geschwistern und setzen des countChildren
105       $this->indizes[$this->path[sizeof($this->path)-1]]["_parent"]["_countChildren"] = $count;
106
107       if ($last>0) {
108         $this->indizes[$last]["_nextBrother"] = &$this->indizes[$this->path[sizeof($this->path)-1]];
109         $this->indizes[$last]["_nextBrother"]["_prevBrother"] =& $this->indizes[$last];
110       }
111       // pfadtiefe adaptieren
112       array_pop($this->path);
113       array_pop($this->path);
114     }
115   }
116
117   #$file = "../../SDE.xml";
118    $file = "../../Rules_SDE_to_ISDE.xml";
119   #$file = "../../Rules_ISDE_to_SYM.xml";
120   #$file = "../../Matrix_CF_from_SYM_to_DIAG.xml";
121   $simple=str_replace(" visible=\"-1\" ","",implode(file($file)));
122   $xxx = new ObjectFromXML($simple);
123   $foo =& $xxx->xml["_firstChild"];
124   $remember_me="";
125
126   do{
127     $bar =& $foo["_firstChild"];
128
129     if ($foo["_name"]=="RULE"){
130       $ri=$foo["_attributes"]["index"];
131       $rule_b[$ri][]="function ".$foo["_lastChild"]["_lastChild"]["_lastChild"]["_firstChild"]["_name"].str_pad(
            $ri,3,"0",STR_PAD_LEFT)."()\n   return ";
132       $rule_e[$ri][]="\nend\n";
133     }elseif($foo["_name"]=="SDE" && $foo["_parent"]["_name"]=="SDE"){
134       $ri=$foo["_attributes"]["index"];
135       $rule_b[$ri][]=$foo["_name"].str_pad($ri,3,"0",STR_PAD_LEFT)."=".$foo["_name"].str_pad($ri,3,"0",
            STR_PAD_LEFT)." or " . $foo["_attributes"]["value"];
136       $rule_e[$ri][]="\n";
137     }elseif($foo["_name"]=="CF"){
138       $ri=str_pad($foo["_firstChild"]["_attributes"]["index"],3,"0",STR_PAD_LEFT).str_pad($foo["_firstChild"]["
            _nextBrother"]["_attributes"]["index"],3,"0",STR_PAD_LEFT);
139       if ($remember_me==str_pad($foo["_firstChild"]["_attributes"]["index"],3,"0",STR_PAD_LEFT)){
140         $rule_b[$ri][]=", _cfinner (".$foo["_firstChild"]["_nextBrother"]["_name"].str_pad($foo["_firstChild"]["
              _nextBrother"]["_attributes"]["index"],3,"0",STR_PAD_LEFT)."(),".$foo["_lastChild"]["_attributes"]["
              value"].")";
141         if ($remember_me!="" && $remember_me!=str_pad($foo["_firstChild"]["_attributes"]["index"],3,"0",
              STR_PAD_LEFT)){
142           $rule_e[$ri][]="})\nend\n";
143         }else $rule_e[$ri][]="";
144       }else{
145         $rule_e[$ri][]="function ".$foo["_name"].str_pad($foo["_firstChild"]["_attributes"]["index"],2,"0",
              STR_PAD_LEFT)." ()\n return _cfouter({_cfinner(".$foo["_firstChild"]["_nextBrother"]["_name"].str_pad(
              $foo["_firstChild"]["_nextBrother"]["_attributes"]["index"],3,"0",STR_PAD_LEFT)."(),".$foo["_lastChild"
              ]["_attributes"]["value"].")";
146         if ($remember_me!="" && $remember_me!=str_pad($foo["_firstChild"]["_attributes"]["index"],3,"0",
              STR_PAD_LEFT)){
147           $rule_b[$ri][]="})\nend\n";
148         }else $rule_b[$ri][]="";
149         $remember_me=str_pad($foo["_firstChild"]["_attributes"]["index"],3,"0",STR_PAD_LEFT);
```

```
150       }
151     }
152
153     do {
154       if (!($bar["_name"]=="THEN" || $bar["_name"]=="ELSE" || $foo["_name"]=="CF" || ($foo["_name"]=="SDE
          " && $foo["_parent"]["_name"]=="SDE"))){
155         // hier beginnt der ungute teil ,.. jetzt koennen alle moeglichen boesen dinge beginnen
156         $rule_b[$ri][]= fill (&$bar);
157       }
158     }while ($bar =& $bar["_nextBrother"]);
159     echo "".implode("",$rule_b[$ri]).implode("",array_reverse($rule_e[$ri]));
160   }while ($foo =& $foo["_nextBrother"]);
161
162   function fill(&$element){
163     global $lookup;
164     $rule=array();
165     $ret="";
166
167     $foo =& $element["_firstChild"];
168     // fuer die cf matrix
169     if ($element["_name"]=="CF") $foo=$element;
170
171     do{
172       $rule[_BEGIN]="";
173       $rule[_END]="";
174       $dont=0;
175
176       switch ($foo["_name"]){
177         case "EXPRESSION":
178           if ($foo["_parent"]["_name"]=="COP") { $rule[_BEGIN]=","; }
179           if ($foo["_parent"]["_name"]=="VOP") { $rule[_BEGIN]="("; $rule[_BEGIN]=")"; }
180           break;
181         case "COP":
182           // wenn der function between an einer stelle eine 0 uebergeben wird,
183           // verhaelt sie sich wie ein groesser oder kleiner, je nach der position der null
184           if (@$foo["_attributes"]["atLeastInclude"]) $atLeast=$foo["_attributes"]["atLeastInclude"];
185           else $atLeast=0;
186           if (@$foo["_attributes"]["atMostInclude"]) $atMost=$foo["_attributes"]["atMostInclude"];
187           else $atMost=0;
188
189           // aenderungen fuer udo, da between nicht immer erwuenscht ist
190           if ($atLeast==0 && $atMost>0){
191             $rule[_BEGIN]="_le(_sum({0";
192             $rule[_END]="}),".$atMost.")";
193           }else if ($atLeast>0 && $atMost==0){
194             $rule[_BEGIN]="_ge(_sum({0";
195             $rule[_END]="}),".$atLeast.")";
196           }else if ($atLeast == $atMost && $atMost>0){
197             $rule[_BEGIN]="_eq(_sum({0";
198             $rule[_END]="}),".$atLeast.")";
199           }else if ($atLeast != $atMost && $atMost>0 && $atLeast>0){
200             $rule[_BEGIN]="_between(_sum({0";
201             $rule[_END]="}),".$atLeast.",".$atMost.",".GREATER_EQUAL.",".LESS_EQUAL.")";
202           }else{
203             die("atLeast und atMost gleich 0");
204           }
```

```
205
206           break;
207         case "LOP":
208           $dont=1;
209           if ( // hier tritt eine between regel der SDE sachen in kraft
210             $foo[" _firstChild "][" _firstChild "]["_name"]=="VOP" && $foo["_firstChild"]["_firstChild"][" _firstChild "
                ]["_name"]=="SDE" && $foo["_lastChild"]["_firstChild"][" _firstChild "]["_name"]=="SDE" && $foo["
                _firstChild"]["_firstChild"][" _firstChild "][" _attributes "]["index"]==$foo["_lastChild"][" _firstChild "][" 
                _firstChild "][" _attributes "]["index"]
211           ){
212             $check[_BEGIN]=$lookup[$foo["_firstChild"][" _firstChild "][" _attributes "]["value" ]];
213             $check[_END] =$lookup[$foo["_lastChild"][" _firstChild "][" _attributes "]["value" ]];
214             $rule[_BEGIN]="_between(".$foo["_firstChild"][" _firstChild "][" _firstChild "]["_name"].str_pad($foo["
                _firstChild "][" _firstChild "][" _firstChild "][" _attributes "]["index" ],3, "0",STR_PAD_LEFT).",".($foo[
                "_firstChild"]["_firstChild "]["_lastChild "][" _attributes "]["value"]) . "," .( $foo[" _lastChild "][" 
                _firstChild "][" _lastChild "][" _attributes "]["value"]+$check[_END]).")";
215           }else $rule[_BEGIN]="_".$foo["_attributes"]["value"]. "({". fill ($foo[" _firstChild "]) . ",". fill ($foo["
                _lastChild "]) . "})";
216           break;
217         case "VOP":
218           if ((
219             $foo[" _firstChild "]["_name"]=="ISDE" && $foo["_attributes"]["value"]=="==" && $foo["_lastChild"]["
                _name"]=="VAL" && $foo["_lastChild"]["_attributes"]["value"]=="1") || ($foo[" _firstChild "]["_name"
                ]=="SDE" && $foo["_attributes"]["value"]=="==" && $foo["_lastChild"]["_name"]=="VAL" && $foo["
                _lastChild"]["_attributes"]["value"]=="1")){
220           }else{
221             if ($foo["_attributes "]["value"]==">="){
222               $rule[_BEGIN]="_ge(";
223               $rule[_END]=",".$foo["_lastChild"][" _attributes "]["value"]. ")";
224             }elseif ($foo["_attributes "]["value"]=="<="){
225               $rule[_BEGIN]="_le(";
226               $rule[_END]=",".$foo["_lastChild"][" _attributes "]["value"]. ")";
227             }elseif ($foo["_attributes "]["value"]=="<"){
228               $rule[_BEGIN]="_lt(";
229               $rule[_END]=",".$foo["_lastChild"][" _attributes "]["value"]. ")";
230             }elseif ($foo["_attributes "]["value"]==">"){
231               $rule[_BEGIN]="_gt(";
232               $rule[_END]=",".$foo["_lastChild"][" _attributes "]["value"]. ")";
233             }elseif ($foo["_attributes "]["value"]=="=="){
234               $rule[_BEGIN]="_equal(";
235               $rule[_END]=",".$foo["_lastChild"][" _attributes "]["value"]. ")";
236             }else{
237               die("value␣in␣". $foo[" _firstChild "]["_name"]."␣oder␣".$foo["_lastChild "]["_name"]."␣ungleich␣1:␣". 
                  $foo[" _lastChild "]["_attributes "]["value"]) ;
238             }
239           }
240           break;
241         case "VAL": break;
242         case "MATH": break;
243         case "MOP":
244           if ($foo["_countChildren"]==2 &&
245             $foo[" _firstChild "]["_name"]=="SDE" && $foo["_lastChild"]["_name"]=="SDE"){
246             $rule[_BEGIN]=$lookup[$foo["_attributes"]["value"]]. "(" .$foo[" _firstChild "]["_name"].$foo[" _firstChild
                "]["_attributes "]["index"]. "," .$foo[" _lastChild "]["_name"].$foo["_lastChild "]["_attributes "]["index"
                ]. ")";
247             $rule[_END]="";
```

```
248             $dont=1;
249           }else die("mop verhaelt sich anders als er sollte");
250           break;
251         case "NEG":
252           $rule[_BEGIN]="_neg(";
253           $rule[_END]=")";
254           break;
255         case "ABS":
256           $rule[_BEGIN]="_abs(";
257           $rule[_END]=")";
258           break;
259         case "SDE":
260           if ($foo["_parent"]["_name"]=="VOP"){
261             $rule[_BEGIN]=$foo["_name"].".".str_pad($foo["_attributes"]["index"],3,"0",STR_PAD_LEFT).".";
262           }
263           break;
264         case "ISDE":
265           if ($foo["_parent"]["_name"]=="VOP"){
266             $rule[_BEGIN]=$foo["_name"].str_pad($foo["_attributes"]["index"],3,"0",STR_PAD_LEFT)."()";
267           }
268           break;
269         case "SYM":
270           echo "sym#";
271           break;
272         default:
273           die("unbekanntes muster: ".($foo["_name"])."\n");
274       }
275       if (@$foo["_countChildren"]>0 && !$dont) $ret=$ret.$rule[_BEGIN].fill(&$foo).$rule[_END];
276       else $ret=$ret.$rule[_BEGIN].$rule[_END];
277     }while ($foo =& $foo["_nextBrother"]);
278     return $ret;
279   }
280   ?>
```

Appendix B: Lua-Inferenzmaschine

Listing B.1: Lua-Funktionen der Inferenz

```
1  if logger == nil then
2      logger = {}
3      function logger:info (text) io.write(text .. "\n") end
4  end
5  local logger = logger
6  local x = x
7
8  function math.round(num, idp)
9    local mult = 10^(idp or 0)
10   return math.floor(num * mult + 0.5) / mult
11 end
12
13 function _and(elements)
14     --elements = elements or return false
15     for key, value in pairs (elements) do
16         if _valueize(value) == 0 then return false end
17     end
18     return true
19 end
20
21 function _or(elements)
22     --elements = elements or return false
23     for key, value in pairs (elements) do
24         if _valueize(value) ~= 0 then return true end
25     end
26     return false
27 end
28
29 function _gt(element1, element2)
30     -- vergleich zweier parameter, ob par1 grösser als par2 ist
31     if not element1 or not element2 then return false end
32     if element1 > element2 then return true
33     else return false
34     end
35 end
36
37 function _ge(element1, element2)
38     -- vergleich zweier parameter, ob par1 grösser_gleich als par2 ist
39     if not element1 or not element2 then return false end
40     if element1 >= element2 then return true
41     else return false
42     end
43 end
44
45 function _lt(element1, element2)
46     -- vergleich zweier parameter, ob par1 kleiner als par2 ist
47     if not element1 or not element2 then return false end
48     if element1 < element2 then return true
49     else return false
50     end
51 end
```

```
52
53 function _le(element1, element2)
54     -- vergleich zweier parameter, ob par1 kleiner_gleich als par2 ist
55     if not element1 or not element2 then return false end
56     if element1 <= element2 then return true
57     else return false
58     end
59 end
60
61
62 function _equal(...)
63     -- vergleicht beliebig viele elemente die in einem table übergeben werden(einziger parameter),
64     -- oder beliebig viele unterschiedliche elemente die als parameter übergeben werden
65     -- wenn ncihts übergeben wird, wird false returned. zwei tables werden auf deren speicherbereiche
66     -- verglichen, tables mit gleichem inhalt aber unterschiedlischen speicherorten sind ungleich!
67     ret=true
68     if not arg then return false end
69     arg2=arg
70     if (arg["n"]<2) then
71         arg2=unpack(arg)
72     end
73
74     for key, value in pairs (arg2) do
75         if key~="n" then if arg2[key]~=arg2[1] then ret=false end end
76     end
77
78     return ret
79 end
80 function _eq(...)
81     -- vergleicht beliebig viele elemente die in einem table übergeben werden(einziger parameter),
82     -- oder beliebig viele unterschiedliche elemente die als parameter übergeben werden
83     -- wenn ncihts übergeben wird, wird false returned. zwei tables werden auf deren speicherbereiche
84     -- verglichen, tables mit gleichem inhalt aber unterschiedlischen speicherorten sind ungleich!
85     ret=true
86     if not arg then return false end
87     arg2=arg
88     if (arg["n"]<2) then
89         arg2=unpack(arg)
90     end
91
92     for key, value in pairs (arg2) do
93         if key~="n" then if arg2[key]~=arg2[1] then ret=false end end
94     end
95
96     return ret
97 end
98
99 function _between(count, lbound, ubound, greater_equal, less_equal) -- works for integer and bool
100     count = count or 1
101     lbound = lbound or 0
102     ubound = ubound or 0
103     greater_equal = _neg(_neg(greater_equal)) or 1  -- für logischen wert (koennte ja > 1 uebergeben
104     less_equal = _neg(_neg(less_equal)) or 0
105
106     if (greater_equal == 1 and count >= lbound) then
107         if (less_equal == 1 and count <= ubound) then
```

```
108             return true
109         elseif (less_equal == 0 and count < ubound) then
110             return true
111         end
112     elseif (greater_equal == 0 and count > lbound) then
113         if (less_equal == 1 and count <= ubound) then
114             return true
115         elseif (less_equal == 0 and count < ubound) then
116             return true
117         end
118     end
119     return false
120 end
121
122
123 function _sum(elements)
124     --elements = elements or return false
125     ret=0
126     for key, value in pairs (elements) do
127         --io.write(_valueize(value));
128         ret=_add(ret,_valueize(value))
129     end
130     return ret
131 end
132
133 function _add(el1, el2) -- for later fuzzyfication
134     return (el1+el2)
135 end
136
137 function _mult(el1, el2) -- for later fuzzyfication
138     if (type(el1)=="number" and type(el2)=="number") then
139         return (el1*el2)
140     else
141         logger:info(el1)
142         logger:info(el2)
143     end
144
145 end
146
147 function _boundmult(el1, el2, lbound, ubound) -- multiplikation mit einer positiven und negativen grenze
148 -- (bei überschrietung der grenze wird der grenzwert als ergebnis geliefert
149     res = 0
150     lbound = lbound or -1
151     ubound = ubound or 1
152     res = _mult(el1, el2)
153     if (res < lbound) then return lbound end
154     if (res > ubound) then return ubound end
155     return res
156 end
157 function _sub(el1,el2) -- for later fuzzyfication
158     return (el1-el2)
159 end
160
161 function _neg(element) -- for later fuzzyfiaction
162     if type(element) == "boolean" then
163         if element == false then return true end
```

```
164         if element == true then return false end
165     else
166         if type(element) == "number" then
167             if element > 0 then return false end
168             if element <= 0 then return true end
169         else
170             if type(element) == "nil" then return true
171             else return false
172             end
173         end
174     end
175 end
176
177 function _abs(element)
178     if type(element) == "string" then
179         return (math.round(tonumber(element))^2)^0.5
180     else
181         return (element^2)^0.5
182     end
183 end
184
185 function _valueize(element)
186     if type(element) == "boolean" then
187         if element == true then return 1
188         else return 0
189         end
190     else
191         if type(element) == "number" then
192             return math.round(element,3)
193         else
194             if type(element) == "string" then
195                 return math.round(tonumber(element))
196             else
197                 if type(element) == "nil" then
198                     return 0
199                 else
200                     return string.format("unknown type `%s'\n", type(element))
201                 end
202             end
203         end
204     end
205 end
206
207 function _cfinner(sym,value) -- for later fuzzyfication
208     if (step==4) then
209 -- leider ist dies die einzig konsistente möglichkeit um die sicherheitsfaktoren nicht zweimal speichern
210 -- zu müssen (step3 und step4). somit wird _cfinner einmal im step3 zum "" der sicherheitsfaktoren verwendet,
211 -- und einmal (wenn step4 gesetzt ist ) um die korrekturfaktoren zu errechnen. hierfür werden die selben
212 -- sicherheitsfaktoren gebraucht -> konsistente speicherung ist wichtiger als korrekte benennung der functions
213         return value
214     else
215         return _mult(_valueize(sym),value)
216     end
217 end
218
219 function _cfouter(elements)
```

```
220     if (step==4) then
221 -- leider ist dies die einzig konsistente möglichkeit um die sicherheitsfaktoren nicht zweimal speichern
222 -- zu müssen (step3 und step4). somit wird _cfouter einmal im step3 zum "addieren" der sicherheitsfaktoren
223 -- verwendet, und einmal (wenn step4 gesetzt ist) um die korrekturfaktoren zu errechnen. hierfür werden
224 -- die selben sicherheitsfaktoren gebraucht -> konsistente speicherung ist wichtiger als korrekte benennung
225 -- der functions :(
226         e=0
227         for i,cfValue in ipairs(elements) do
228             if (cfValue > 0) then
229                 a=0;
230                 b=0;
231                 c=0;
232                 d=0;
233
234                 a = e;
235                 b = cfValue;
236                 c = a + b;
237                 d = a * b;
238                 e = c - d;
239             end
240         end
241
242         if e ~= 0 then
243             e = 1 / e
244         end
245         return e
246     else
247         diagValue = 0
248 -- eigentlich geplanter "standardfall für diese function
249 -- jedes element; und somit jedes symptom zur entsprechenden diagnose durchgehen
250 -- da die symptome mit ihrem sicherheitsfaktor multiplizert sind, muss nur auf deren ungleichheit mit null
251 -- prüfen zum glück verändern sicherheitsfaktoren mit 0 bei einem symptom mit 1 den diagValue-wert nicht
252 -- sonst wäre das sehr schlecht für das cfinner/cfouter prinzip.
253         for i,cfValue in ipairs(elements) do
254             caseNumber = 0
255             if cfValue ~= 0 then -- es gibt ein symptom mit einem sicherheitsfaktor
256
257                 a = 0 --// help variable for switch-case
258                 b = 0 --// help variable for switch-case
259                 c = 0 --// help variable for switch-case -> newly calulated certainty factor of diagnosis
260
261                 if ((diagValue > 0) and (cfValue > 0))     then caseNumber=1 -- both certainty factors are pos
262                     a = diagValue + cfValue;
263                     b = diagValue * cfValue;
264                     c = a - b;
265                 elseif ((diagValue < 0) and (cfValue < 0)) then caseNumber=2 -- both certainty factors are
                      neg
266                     a = diagValue + cfValue;
267                     b = diagValue * cfValue;
268                     c = a + b;
269                 else caseNumber=3 -- both certainty factors are 0 or they have different algebraic signs
270                     a = diagValue + cfValue;
271                     b = 1 - math.min(math.abs(diagValue), math.abs(cfValue));
272                     c = a / b;
273                 end
274
```

```
275                 --if ( type(c) == nil ) then diagValue=-1
276                 --else                       diagValue=c
277                 --end
278     --              logger:info(i .. " " .. c)
279                 diagValue=c
280             end
281         end
282         return diagValue
283     end
284 end
285
286 function doscript (filename)
287 -----------------------------------------
288 -- Execute a script
289 -- If an error is found, Lua's error handler is called and this function
290 -- does not return
291 -----------------------------------------
292         local res, err = loadfile(filename)
293         if not res then
294                 error (format ("Cannot␣execute␣`%s'.␣Exiting.\n%s", filename, err))
295         else
296                 return res ()
297         end
298 end
299
300 function doif (filename)
301 -----------------------------------------
302 -- Execute the file if there is no "file error".
303 -- If an error is found, and it is not a "file error", Lua 'error'
304 -- is called and this function does not return
305 -----------------------------------------
306         if not filename then return end  -- no file
307         local f, err = open(filename)
308         if not f then return nil, err end    -- no file (or unreadable file)
309         f:close()
310         return doscript (filename)
311 end
312
313 function print_r (t, indent, done)
314     done = done or {}
315     indent = indent or 0
316     if type(t) == "table" then
317         for key, value in pairs (t) do
318             foo=(string.rep ("␣", indent)) -- indent it
319             if type (value) == "table" and not done [value] then
320               done [value] = true
321               logger:info(foo .. string.format("[%s]␣=>␣table", tostring (key)));
322               logger:info(string.rep ("␣", indent+4) .. "(") -- indent it
323               --logger:info("(\n");
324               print_r (value, indent + 7, done)
325               logger:info(string.rep ("␣", indent+4) .. ")") -- indent it
326               --logger:info(")\n");
327             else
328               logger:info(foo .. string.format("[%s]␣=>␣%s", tostring (key),value))
329             end
330         end
```

```
331     else
332         if type(t) == "nil" then
333             logger:info(string.format("%s",type(t)))
334         else
335             logger:info(string.format("%s",_valueize(t)))
336         end
337     end
338 end
339
340 function print_xml (t, indent, done)
341     done = done or {}
342     indent = indent or 0
343     if type(t) == "table" then
344         for key, value in pairs (t) do
345             foo=(string.rep (" ", indent)) -- indent it
346             if type (value) == "table" and not done [value] then
347               done [value] = true
348               logger:info(foo .. string.format("[%s] => table", tostring (key)));
349               logger:info(string.rep (" ", indent+4) .. "(") -- indent it
350               --logger:info("(\n");
351               print_r (value, indent + 7, done)
352               logger:info(string.rep (" ", indent+4) .. ")") -- indent it
353               --logger:info(")\n");
354             else
355               logger:info(foo .. string.format("[%s] => %s", tostring (key),value))
356             end
357         end
358     else
359         if type(t) == "nil" then
360             logger:info(string.format("%s",type(t)))
361         else
362             logger:info(string.format("%s",_valueize(t)))
363         end
364     end
365 end
366
367
368 function positives (...)
369     ret={}
370     if not arg then return false end
371     arg2=arg
372     if (arg["n"]<2) then
373         arg2=unpack(arg)
374     end
375
376     for key, value in pairs (arg2) do
377         if value >= 0 then
378             ret[key]=value
379         end
380     end
381
382     return ret
383 end
384
385 function negatives(...)
386     ret={}
```

```
387     if not arg then return false end
388     arg2=arg
389     if (arg["n"]<2) then
390         arg2=unpack(arg)
391     end
392
393     for key, value in pairs (arg2) do
394         if value <= 0 then
395             ret[key]=_abs(value)
396         end
397     end
398
399     return ret
400 end
401
402 function sortIt(rray,hunni)
403     keys={}
404     vals={[1]=1000}
405     hunni = hunni or 100
406     for k, v in pairs(rray) do
407         v=_mult(v,hunni);
408         for i=1, table.getn(vals), 1 do
409             if v<vals[i] then
410                 table.insert(vals,i,v)
411                 table.insert(keys,i,k)
412                 break
413             end
414         end
415     end
416     table.remove(vals,table.getn(vals))
417
418     return {keys = keys,vals = vals}
419 end
420
421 function map(func, array)
422   local new_array = {}
423   for i,v in ipairs(array) do
424     new_array[i] = func(v)
425   end
426   return new_array
427 end
428
429 function foreach(elements, func)
430     table.foreach(elements,func)
431     return elements
432 end
433
434 function variance(elements, border)
435     foo=0
436     foo_mean=0
437     for i=1,border,1 do foo_mean=foo_mean+elements.vals[i] end
438     foo_mean=foo_mean/border
439     for i=1,border,1 do foo=foo+(foo_mean-elements.vals[i])^2 end
440     foo=foo^0.5
441     --print_r(foo_mean)
442
```

```
443     bar=0
444     bar_mean=0
445     for i=(border+1),table.getn(elements.vals) do bar_mean=bar_mean+elements.vals[i] end
446     bar_mean=bar_mean/(table.getn(elements.vals)-border)
447     for i=(border+1),table.getn(elements.vals) do bar=bar+(bar_mean-elements.vals[i])^2 end
448     bar=bar^0.5
449
450     --print(foo .. " @ " .. foo_mean .. " vs " .. bar .. " @ " .. bar_mean .. " eq " .. (((foo-bar)^2)^0.5) .. "
          or " .. (foo+bar))
451     return (((foo-bar)^2)^0.5)
452     --return (foo+bar)
453 end
454
455 --
456 --print_r(_between(1,0,0));
457
458
459 function step5_gradient(elements) -- gradientenverfahren
460     vari={}; visible={pos_vals={},pos_keys={},neg_vals={},neg_keys={}}
461
462     -- aufsplitten der positiven und negativen werte
463     pos=positives(elements); neg=negatives(elements)
464     pos=sortIt(pos,1)                ;neg=sortIt(neg,1)
465
466     -- dynamischer schwellwert:
467     table.insert(pos.vals,1,0)       ;table.insert(neg.vals,1,0)       -- stützstelle im minimum bei 0
468     table.insert(pos.vals,1)         ;table.insert(neg.vals,1)         -- stützstelle im maximum bei 1
469     table.insert(pos.keys,1,"low") ;table.insert(neg.keys,1,"low"); -- beschriftung der stützstelle
470     table.insert(pos.keys,"high")  ;table.insert(neg.keys,"high") ; -- beschriftung der stützstelle
471
472     -- positiver teil (hinweisend)
473     -- alle möglichen varianzen durchgehen und danach sortieren
474     maxx={}
475     maxx.diff=0
476
477     for i=table.getn(pos.vals)-1, 1, -1 do
478         foo=pos.vals[i+1]-pos.vals[i]
479         if foo > maxx.diff then
480             maxx.diff=foo
481             maxx.index=i
482         end
483         --print_r(pos.vals[i])
484     end
485
486     print_r(pos)
487     print_r(maxx)
488
489 if true==false then
490     vari={}
491     -- negativer teil (wegweisend)
492     -- alle möglichen varianzen durchgehen und danach sortieren
493     for i=1, table.getn(neg.vals)-1, 1 do vari[i]=variance(neg,i) end
494     vari=sortIt(vari,1)
495
496     -- das erste der sortierten varianz-elemente ist das ergebnis wo der schwellwert zu setzen ist, array mit
        anzuzeigenden werten befüllen
```

```
497     for i=(vari.keys[1]+1),table.getn(neg.vals),1 do
498         table.insert( visible ["neg_vals"], neg.vals[i]/100)
499         table.insert( visible ["neg_keys"], neg.keys[i])
500     end
501     -- hilfsstellen rausnehmen
502     table.remove (visible["neg_vals"], table.getn( visible ["neg_vals"]) )
503     table.remove (visible["neg_keys"], table.getn( visible ["neg_keys"]) )
504 end
505     --print_r( visible )
506     --return visible
507 end
508
509
510
511
512
513
514
515
516 path="H:/20070112/nun"
517
518 doscript(path .. "/umwandlung/step0.lua")
519 doscript(path .. "/umwandlung/step1.lua")
520 doscript(path .. "/umwandlung/step2.lua")
521 doscript(path .. "/umwandlung/step3.lua")
522 doscript(path .. "/umwandlung/step4.lua")
523 doscript(path .. "/umwandlung/step5.lua")
524
525 SDE021=SDE021 or 1
526 SDE086=SDE086 or 1
527 SDE598=SDE598 or 30
528 SDE616=SDE616 or 1
529 SDE617=SDE617 or 70.0
530
531
532 --print_r(foo)
533 --print_r(_between(SDE599,-1,5))
534 --elements={DIAG01 = CF01(), DIAG02 = CF02(), DIAG03 = CF03(), DIAG04 = CF04(), DIAG05 = CF05(),
    DIAG06 = CF06(), DIAG07 = CF07(), DIAG08 = CF08(), DIAG09 = CF09(), DIAG10 = CF10(), DIAG11 =
    CF11(), DIAG12 = CF12(), DIAG13 = CF13(), DIAG14 = CF14(), DIAG15 = CF15(), DIAG16 = CF16(),
    DIAG17 = CF17(), DIAG18 = CF18(), DIAG19 = CF19(), DIAG20 = CF20()}
535 elements={CF01(),CF02(),CF03(),CF04(),CF05(),CF06(),CF07(),CF08(),CF09(),CF10(),CF11(),CF12(),CF13(),
    CF14(),CF15(),CF16(),CF17(),CF18(),CF19(),CF20()}
536 step=4
537
538 --print_r(elements)
539 --x=(step4(elements));
540 --print_r(x);
541 x={0.73000000258454,0.55100231307039,0.55322540093059}
542 --x=step5_gradient(step4(elements));
543 x=step5(x)
544 print_r(x);
545
546
547 --print_r(_cfinner (0,0.3) )
548 --print_r(cfs [20]() )
```

```
549 --print_r(_cfinner (0,0.3) )
550 --print_r(CF20())
551
552 if true == false then -- andere art zum ausklammern von code ;)
553 io.write("" .. _cfinner(-0.000001, 0.3) .. "\n");
554 io.write("" .. _cfinner(-0.000001, 0.0) .. "\n");
555 io.write("" .. _cfinner(1,  0.0) .. "\n");
556 io.write("" .. _cfinner(1,  0.3) .. "\n");
557
558 io.write("" ..  _valueize(_or({-0.000001,-0.000001,0})) .. "\n")
559 io.write("" ..  _valueize(_or({-0.000001,1,1})) .. "\n")
560 io.write("" ..  _valueize(_or({0,1,1})) .. "\n")
561 io.write("" ..  _valueize(_or({1,1,1})) .. "\n")
562 --io.write("" .. _sum({0,-0.000001,-0.000001,-0.000001,1,1}) .. "\n")
563
564 end
565
566 if true == false then
567 io.write("between -1.5 0,4:" .. _valueize(_between(-1.5,0,4)) .. "\n")
568 io.write("between -0.5 0,4:" .. _valueize(_between(-0.5,0,4)) .. "\n")
569 io.write("between 0.5 0,4:" .. _valueize(_between(0.5,0,4)) .. "\n")
570 io.write("between 1.5 0,4:" .. _valueize(_between(1.5,0,4)) .. "\n")
571 io.write("between 3.5 0,4:" .. _valueize(_between(3.5,0,4)) .. "\n")
572 io.write("between 4.5 0,4:" .. _valueize(_between(4.5,0,4)) .. "\n")
573 io.write("between 5.5 0,4:" .. _valueize(_between(5.5,0,4)) .. "\n")
574
575
576
577
578
579 SDE001 = 0.0000000000000000001
580 SDE002 = 0
581 SDE003 = nan
582
583 if (SDE001 > 0) then
584     io.write("grösser")
585 else
586     io.write("nicht grösser")
587 end
588 io.write("\n")
589
590 if (SDE001 < 0) then
591     io.write("kleiner")
592 else
593     io.write("nicht kleiner")
594 end
595 io.write("\n")
596 if (SDE001 == 0) then
597     io.write("gleich")
598 else
599     io.write("nicht gleich")
600 end
601 io.write("\n")
602
603
604
```

```
605 io.write(type(SDE001) .. math.round(SDE001))
606 io.write(type(SDE002))
607 io.write(type(SDE003))
608 end
609
610 --print_r(elements)
611 --
612
613 --logger:info("juhu")
614 -- tests if object Statement is nil
615 --if x == nil then
616     --logger:info("Error. x object is nil")
617     --return
618 --end
619 --[[
620
621 print_r("############################")
622 print_r(x);
623 print_r("############################")
624
625 SDE1=SDE001 or -1
626 SDE2=SDE002 or -1
627 SDE3=SDE003 or -1
628 SDE4=SDE004 or -1
629 SDE5=SDE005 or -1
630 SDE6=SDE616 or -1
631 SDE7=SDE617 or -1
632 SDE8=SDE008 or -1
633 SDE9=SDE009 or -1
634
635 io.write("#")
636 print_r(SDE616)
637 print_r(ISDE623())
638 print_r(ISDE624())
639 print_r(ISDE625())
640 print_r(ISDE626())
641 ]]--
642
643 --[[
644 print_r(_between(4,-1,5))
645 ]]
646
647 if true == false then
648 SDE={}
649 for i=1,10 do SDE[i]=1 end
650 for i=11,700 do SDE[i]=0 end
651
652 elements={DIAG01 = CF01(), DIAG02 = CF02(), DIAG03 = CF03(), DIAG04 = CF04(), DIAG05 = CF05(),
      DIAG06 = CF06(), DIAG07 = CF07(), DIAG08 = CF08(), DIAG09 = CF09(), DIAG10 = CF10(), DIAG11 =
      CF11(), DIAG12 = CF12(), DIAG13 = CF13(), DIAG14 = CF14(), DIAG15 = CF15(), DIAG16 = CF16(),
      DIAG17 = CF17(), DIAG18 = CF18(), DIAG19 = CF19(), DIAG20 = CF20()}
653 x=(step4(elements));
654 print_r(x);
655
656 end
```

Appendix C: Neue Lua-Regeln

Für Listing C.1 sei vermerkt, dass der angeführte Code nicht kompilieren würde. Die Kommentarzeilen mit Inhalten wie `--[[ Rule 1: --]]` wären im originalen Quelltext durch den jeweiligen Funktionsnamen des Symptoms (mit der entsprechenden Indexposition) dargestellt. Aus Gründen der Lesbarkeit wurde davon aber Abstand genommen.

Listing C.1: Lua-Quelltext der Regeln des zweiten Schrittes

```
1  --[[    RULE 1:    --]]
2  ISDE001()
3
4  --[[    RULE 2:    --]]
5  ISDE002()
6
7  --[[    RULE 3:    --]]
8  ISDE003()
9
10 --[[    RULE 4:    --]]
11 ISDE004()
12
13 --[[    RULE 5:    --]]
14 ISDE005()
15
16 --[[    RULE 6:    --]]
17 ISDE006()
18
19 --[[    RULE 7:    --]]
20 ISDE007()
21
22 --[[    RULE 8:    --]]
23 ISDE008()
24
25 --[[    RULE 9:    --]]
26 ISDE009()
27
28 --[[    RULE 10:    --]]
29 ISDE010()
30
31 --[[    RULE 11:    --]]
32 ISDE011()
33
34 --[[    RULE 12:    --]]
35 ISDE012()
36
37 --[[    RULE 13:    --]]
38 ISDE013()
39
40 --[[    RULE 14:    --]]
41 ISDE014()
42
43 --[[    RULE 15:    --]]
```

```
44 ISDE015()
45
46 --[[    RULE 16:    --]]
47 ISDE016()
48
49 --[[    RULE 17:    --]]
50 ISDE017()
51
52 --[[    RULE 18:    --]]
53 ISDE018()
54
55 --[[    RULE 19:    --]]
56 ISDE019()
57
58 --[[    RULE 20:    --]]
59 ISDE020()
60
61 --[[    RULE 21:    --]]
62 _neg(
63   _ge(
64     _sum(
65       {0,ISDE021(),ISDE022(),ISDE023(),ISDE024(),ISDE025(),ISDE026(),ISDE027(),ISDE028(),ISDE029(),
         ISDE030(),ISDE031(),ISDE032(),ISDE033(),ISDE034(),ISDE035(),ISDE036(),ISDE044(),ISDE045(),
         ISDE046(),ISDE047(),ISDE048(),ISDE049(),ISDE050(),ISDE051(),ISDE052(),ISDE053(),ISDE054(),
         ISDE055(),ISDE056(),ISDE057(),ISDE058(),ISDE059(),ISDE060(),ISDE061(),ISDE062(),ISDE063(),
         ISDE064(),ISDE065(),ISDE066(),ISDE067(),ISDE068(),ISDE069(),ISDE070(),ISDE071(),ISDE072(),
         ISDE073(),ISDE074(),ISDE075(),ISDE076(),ISDE077(),ISDE078(),ISDE079(),ISDE080(),ISDE081(),
         ISDE082(),ISDE083(),ISDE084(),ISDE085(),ISDE086(),ISDE087(),ISDE088(),ISDE089(),ISDE090(),
         ISDE091(),ISDE092(),ISDE093(),ISDE094(),ISDE095(),ISDE096(),ISDE097(),ISDE098(),ISDE099(),
         ISDE100(),ISDE101(),ISDE102(),ISDE103(),ISDE104(),ISDE105(),ISDE106(),ISDE107(),ISDE108(),
         ISDE109(),ISDE110(),ISDE111(),ISDE112(),ISDE113(),ISDE114(),ISDE115(),ISDE116(),ISDE117(),
         ISDE118(),ISDE119(),ISDE120(),ISDE121(),ISDE122(),ISDE123(),ISDE124(),ISDE125(),ISDE126(),
         ISDE127(),ISDE128(),ISDE129(),ISDE130(),ISDE131(),ISDE132(),ISDE133(),ISDE134(),ISDE135(),
         ISDE136(),ISDE137(),ISDE138(),ISDE139(),ISDE140(),ISDE141(),ISDE142(),ISDE143(),ISDE144(),
         ISDE145(),ISDE146(),ISDE147(),ISDE148(),ISDE149(),ISDE150(),ISDE151(),ISDE152(),ISDE153(),
         ISDE154(),ISDE155(),ISDE156(),ISDE157(),ISDE158(),ISDE159(),ISDE160(),ISDE161(),ISDE162(),
         ISDE163(),ISDE164(),ISDE165(),ISDE166(),ISDE167(),ISDE168(),ISDE169(),ISDE170(),ISDE171(),
         ISDE172(),ISDE173(),ISDE174(),ISDE175(),ISDE176(),ISDE177(),ISDE178(),ISDE179(),ISDE180(),
         ISDE181(),ISDE182(),ISDE183(),ISDE184(),ISDE185(),ISDE186(),ISDE187(),ISDE188(),ISDE189(),
         ISDE190(),ISDE191(),ISDE192(),ISDE193(),ISDE194(),ISDE195(),ISDE196(),ISDE197(),ISDE198(),
         ISDE199(),ISDE200(),ISDE201(),ISDE202(),ISDE203(),ISDE204(),ISDE205(),ISDE206(),ISDE207(),
         ISDE208(),ISDE209(),ISDE210(),ISDE211(),ISDE212(),ISDE213(),ISDE214(),ISDE215(),ISDE216(),
         ISDE217(),ISDE218(),ISDE219(),ISDE220(),ISDE221(),ISDE222(),ISDE223(),ISDE224(),ISDE225(),
         ISDE226(),ISDE227(),ISDE228(),ISDE229(),ISDE230(),ISDE231(),ISDE232(),ISDE233(),ISDE234(),
         ISDE235(),ISDE236(),ISDE237(),ISDE238(),ISDE239(),ISDE240(),ISDE241(),ISDE242(),ISDE243(),
         ISDE244(),ISDE245(),ISDE246(),ISDE247(),ISDE248(),ISDE249(),ISDE250(),ISDE251(),ISDE252(),
         ISDE253(),ISDE254(),ISDE255(),ISDE256(),ISDE257(),ISDE258(),ISDE259(),ISDE260(),ISDE261(),
         ISDE262(),ISDE263(),ISDE264(),ISDE265(),ISDE266(),ISDE267(),ISDE268(),ISDE269(),ISDE270(),
         ISDE271(),ISDE272(),ISDE273(),ISDE274(),ISDE275(),ISDE276(),ISDE277(),ISDE278(),ISDE279(),
         ISDE280(),ISDE281(),ISDE282(),ISDE283(),ISDE284(),ISDE285(),ISDE286(),ISDE287(),ISDE288(),
         ISDE289(),ISDE290(),ISDE291(),ISDE292(),ISDE293(),ISDE294(),ISDE295(),ISDE296(),ISDE297(),
         ISDE298(),ISDE299(),ISDE300(),ISDE301(),ISDE302(),ISDE303(),ISDE304(),ISDE305(),ISDE306(),
         ISDE307(),ISDE308(),ISDE309(),ISDE310(),ISDE311(),ISDE312(),ISDE313(),ISDE314(),ISDE315(),
         ISDE316(),ISDE317(),ISDE318(),ISDE319(),ISDE320(),ISDE321(),ISDE322(),ISDE323(),ISDE324(),
         ISDE325(),ISDE326(),ISDE327(),ISDE328(),ISDE329(),ISDE330(),ISDE331(),ISDE332(),ISDE333(),
         ISDE334(),ISDE335(),ISDE336(),ISDE337(),ISDE338(),ISDE339(),ISDE340(),ISDE341(),ISDE342(),
```

```
        ISDE343(),ISDE344(),ISDE345(),ISDE346(),ISDE347(),ISDE362(),ISDE363(),ISDE364(),ISDE365(),
        ISDE366(),ISDE367(),ISDE368(),ISDE369(),ISDE370(),ISDE371(),ISDE372(),ISDE373(),ISDE374(),
        ISDE375(),ISDE376(),ISDE377(),ISDE378(),ISDE379(),ISDE380(),ISDE381(),ISDE382(),ISDE383(),
        ISDE384(),ISDE385(),ISDE386(),ISDE387(),ISDE388(),ISDE389(),ISDE390(),ISDE391(),ISDE392(),
        ISDE393(),ISDE394(),ISDE395(),ISDE396(),ISDE397(),ISDE398(),ISDE399(),ISDE400(),ISDE401(),
        ISDE402(),ISDE403(),ISDE404(),ISDE405(),ISDE406(),ISDE407(),ISDE408(),ISDE409(),ISDE410(),
        ISDE411(),ISDE412(),ISDE413(),ISDE414(),ISDE415(),ISDE416(),ISDE417(),ISDE418(),ISDE419(),
        ISDE420(),ISDE421(),ISDE422(),ISDE423(),ISDE424(),ISDE425(),ISDE426(),ISDE427(),ISDE428(),
        ISDE429(),ISDE430(),ISDE431(),ISDE432(),ISDE433(),ISDE434(),ISDE435(),ISDE436(),ISDE437(),
        ISDE438(),ISDE439(),ISDE440(),ISDE441(),ISDE442(),ISDE443(),ISDE444(),ISDE445(),ISDE446(),
        ISDE447(),ISDE448(),ISDE449(),ISDE450(),ISDE451(),ISDE452(),ISDE453(),ISDE454(),ISDE455(),
        ISDE456(),ISDE457(),ISDE458(),ISDE459(),ISDE460(),ISDE461(),ISDE462(),ISDE463(),ISDE464(),
        ISDE465(),ISDE466(),ISDE467(),ISDE468(),ISDE469(),ISDE470(),ISDE471(),ISDE472(),ISDE473(),
        ISDE474(),ISDE475(),ISDE476(),ISDE477(),ISDE478(),ISDE479(),ISDE480(),ISDE481(),ISDE482(),
        ISDE483(),ISDE484(),ISDE485(),ISDE486(),ISDE487(),ISDE488(),ISDE489(),ISDE490(),ISDE491(),
        ISDE492(),ISDE493(),ISDE494(),ISDE495(),ISDE496(),ISDE497(),ISDE498(),ISDE499(),ISDE500(),
        ISDE501(),ISDE502(),ISDE503(),ISDE504(),ISDE505(),ISDE506(),ISDE507(),ISDE508(),ISDE509(),
        ISDE510(),ISDE511(),ISDE512(),ISDE513(),ISDE514(),ISDE515(),ISDE516(),ISDE517(),ISDE518(),
        ISDE519(),ISDE520(),ISDE521(),ISDE522(),ISDE523(),ISDE524(),ISDE525(),ISDE526(),ISDE527(),
        ISDE528(),ISDE529(),ISDE530(),ISDE531(),ISDE532(),ISDE533(),ISDE534(),ISDE535(),ISDE536(),
        ISDE537(),ISDE538(),ISDE539(),ISDE540(),ISDE541(),ISDE542(),ISDE543(),ISDE544(),ISDE545(),
        ISDE546(),ISDE547(),ISDE548(),ISDE549(),ISDE550(),ISDE551(),ISDE552(),ISDE553(),ISDE554(),
        ISDE555(),ISDE556(),ISDE557(),ISDE558(),ISDE559(),ISDE560(),ISDE561(),ISDE562(),ISDE563(),
        ISDE564(),ISDE565(),ISDE566(),ISDE567(),ISDE568(),ISDE569(),ISDE570(),ISDE571(),ISDE572(),
        ISDE573(),ISDE574(),ISDE575(),ISDE576(),ISDE577(),ISDE578(),ISDE579(),ISDE580(),ISDE581(),
        ISDE582(),ISDE583(),ISDE584(),ISDE585()}
66      ),1
67    )
68  )
69
70  --[[     RULE 22:    --]]
71  _ge(
72    _sum(
73      {0,ISDE021(),ISDE029(),ISDE044(),ISDE052(),ISDE060(),ISDE068(),ISDE076(),ISDE084(),ISDE092(),
        ISDE100(),ISDE108(),ISDE116(),ISDE124(),ISDE132(),ISDE140(),ISDE148(),ISDE156(),ISDE164(),
        ISDE172(),ISDE180(),ISDE188(),ISDE196(),ISDE204(),ISDE212(),ISDE220(),ISDE228(),ISDE236(),
        ISDE244(),ISDE252(),ISDE260(),ISDE268(),ISDE276(),ISDE284(),ISDE292(),ISDE300(),ISDE308(),
        ISDE316(),ISDE324(),ISDE332(),ISDE340(),ISDE362(),ISDE370(),ISDE378(),ISDE386(),ISDE394(),
        ISDE402(),ISDE410(),ISDE418(),ISDE426(),ISDE434(),ISDE442(),ISDE450(),ISDE458(),ISDE466(),
        ISDE474(),ISDE482(),ISDE490(),ISDE498(),ISDE506(),ISDE514(),ISDE522(),ISDE530(),ISDE538(),
        ISDE546(),ISDE554(),ISDE562(),ISDE570(),ISDE578()}
74    ),1
75  )
76
77  --[[     RULE 23:    --]]
78  _ge(
79    _sum(
80      {0,ISDE022(),ISDE030(),ISDE045(),ISDE053(),ISDE061(),ISDE069(),ISDE077(),ISDE085(),ISDE093(),
        ISDE101(),ISDE109(),ISDE117(),ISDE125(),ISDE133(),ISDE141(),ISDE149(),ISDE157(),ISDE165(),
        ISDE173(),ISDE181(),ISDE189(),ISDE197(),ISDE205(),ISDE213(),ISDE221(),ISDE229(),ISDE237(),
        ISDE245(),ISDE253(),ISDE261(),ISDE269(),ISDE277(),ISDE285(),ISDE293(),ISDE301(),ISDE309(),
        ISDE317(),ISDE325(),ISDE333(),ISDE341(),ISDE363(),ISDE371(),ISDE379(),ISDE387(),ISDE395(),
        ISDE403(),ISDE411(),ISDE419(),ISDE427(),ISDE435(),ISDE443(),ISDE451(),ISDE459(),ISDE467(),
        ISDE475(),ISDE483(),ISDE491(),ISDE499(),ISDE507(),ISDE515(),ISDE523(),ISDE531(),ISDE539(),
        ISDE547(),ISDE555(),ISDE563(),ISDE571(),ISDE579()}
81    ),1
```

```
82 )
83
84 --[[    RULE 24:    --]]
85 _ge(
86   _sum(
87     {0,ISDE023(),ISDE031(),ISDE046(),ISDE054(),ISDE062(),ISDE070(),ISDE078(),ISDE086(),ISDE094(),
       ISDE102(),ISDE110(),ISDE118(),ISDE126(),ISDE134(),ISDE142(),ISDE150(),ISDE158(),ISDE166(),
       ISDE174(),ISDE182(),ISDE190(),ISDE198(),ISDE206(),ISDE214(),ISDE222(),ISDE230(),ISDE238(),
       ISDE246(),ISDE254(),ISDE262(),ISDE270(),ISDE278(),ISDE286(),ISDE294(),ISDE302(),ISDE310(),
       ISDE318(),ISDE326(),ISDE334(),ISDE342(),ISDE364(),ISDE372(),ISDE380(),ISDE388(),ISDE396(),
       ISDE404(),ISDE412(),ISDE420(),ISDE428(),ISDE436(),ISDE444(),ISDE452(),ISDE460(),ISDE468(),
       ISDE476(),ISDE484(),ISDE492(),ISDE500(),ISDE508(),ISDE516(),ISDE524(),ISDE532(),ISDE540(),
       ISDE548(),ISDE556(),ISDE564(),ISDE572(),ISDE580()}
88   ),1
89 )
90
91 --[[    RULE 25:    --]]
92 _ge(
93   _sum(
94     {0,ISDE024(),ISDE032(),ISDE047(),ISDE055(),ISDE063(),ISDE071(),ISDE079(),ISDE087(),ISDE095(),
       ISDE103(),ISDE111(),ISDE119(),ISDE127(),ISDE135(),ISDE143(),ISDE151(),ISDE159(),ISDE167(),
       ISDE175(),ISDE183(),ISDE191(),ISDE199(),ISDE207(),ISDE215(),ISDE223(),ISDE231(),ISDE239(),
       ISDE247(),ISDE255(),ISDE263(),ISDE271(),ISDE279(),ISDE287(),ISDE295(),ISDE303(),ISDE311(),
       ISDE319(),ISDE327(),ISDE335(),ISDE343(),ISDE365(),ISDE373(),ISDE381(),ISDE389(),ISDE397(),
       ISDE405(),ISDE413(),ISDE421(),ISDE429(),ISDE437(),ISDE445(),ISDE453(),ISDE461(),ISDE469(),
       ISDE477(),ISDE485(),ISDE493(),ISDE501(),ISDE509(),ISDE517(),ISDE525(),ISDE533(),ISDE541(),
       ISDE549(),ISDE557(),ISDE565(),ISDE573(),ISDE581()}
95   ),1
96 )
97
98 --[[    RULE 26:    --]]
99 _eq(
100   _sum(
101     {0,_or(
102       {ISDE025(),ISDE026()}
103     ),_or(
104       {ISDE033(),ISDE034()}
105     ),_or(
106       {ISDE048(),ISDE049()}
107     ),_or(
108       {ISDE056(),ISDE057()}
109     ),_or(
110       {ISDE064(),ISDE065()}
111     ),_or(
112       {ISDE072(),ISDE073()}
113     ),_or(
114       {ISDE080(),ISDE081()}
115     ),_or(
116       {ISDE088(),ISDE089()}
117     ),_or(
118       {ISDE096(),ISDE097()}
119     ),_or(
120       {ISDE104(),ISDE105()}
121     ),_or(
122       {ISDE112(),ISDE113()}
123     ),_or(
```

```
124         {ISDE120(),ISDE121()}
125       ),_or(
126         {ISDE128(),ISDE129()}
127       ),_or(
128         {ISDE136(),ISDE137()}
129       ),_or(
130         {ISDE144(),ISDE145()}
131       ),_or(
132         {ISDE152(),ISDE153()}
133       ),_or(
134         {ISDE160(),ISDE161()}
135       ),_or(
136         {ISDE168(),ISDE169()}
137       ),_or(
138         {ISDE176(),ISDE177()}
139       ),_or(
140         {ISDE184(),ISDE185()}
141       ),_or(
142         {ISDE192(),ISDE193()}
143       ),_or(
144         {ISDE200(),ISDE201()}
145       ),_or(
146         {ISDE208(),ISDE209()}
147       ),_or(
148         {ISDE216(),ISDE217()}
149       ),_or(
150         {ISDE224(),ISDE225()}
151       ),_or(
152         {ISDE232(),ISDE233()}
153       ),_or(
154         {ISDE240(),ISDE241()}
155       ),_or(
156         {ISDE248(),ISDE249()}
157       ),_or(
158         {ISDE256(),ISDE257()}
159       ),_or(
160         {ISDE264(),ISDE265()}
161       ),_or(
162         {ISDE272(),ISDE273()}
163       ),_or(
164         {ISDE280(),ISDE281()}
165       ),_or(
166         {ISDE288(),ISDE289()}
167       ),_or(
168         {ISDE296(),ISDE297()}
169       ),_or(
170         {ISDE304(),ISDE305()}
171       ),_or(
172         {ISDE312(),ISDE313()}
173       ),_or(
174         {ISDE320(),ISDE321()}
175       ),_or(
176         {ISDE328(),ISDE329()}
177       ),_or(
178         {ISDE336(),ISDE337()}
179       ),_or(
```

```
180         {ISDE344(),ISDE345()}
181       ),_or(
182         {ISDE366(),ISDE367()}
183       ),_or(
184         {ISDE374(),ISDE375()}
185       ),_or(
186         {ISDE382(),ISDE383()}
187       ),_or(
188         {ISDE390(),ISDE391()}
189       ),_or(
190         {ISDE398(),ISDE399()}
191       ),_or(
192         {ISDE406(),ISDE407()}
193       ),_or(
194         {ISDE414(),ISDE415()}
195       ),_or(
196         {ISDE422(),ISDE423()}
197       ),_or(
198         {ISDE430(),ISDE431()}
199       ),_or(
200         {ISDE438(),ISDE439()}
201       ),_or(
202         {ISDE446(),ISDE447()}
203       ),_or(
204         {ISDE454(),ISDE455()}
205       ),_or(
206         {ISDE462(),ISDE463()}
207       ),_or(
208         {ISDE470(),ISDE471()}
209       ),_or(
210         {ISDE478(),ISDE479()}
211       ),_or(
212         {ISDE486(),ISDE487()}
213       ),_or(
214         {ISDE494(),ISDE495()}
215       ),_or(
216         {ISDE502(),ISDE503()}
217       ),_or(
218         {ISDE510(),ISDE511()}
219       ),_or(
220         {ISDE518(),ISDE519()}
221       ),_or(
222         {ISDE526(),ISDE527()}
223       ),_or(
224         {ISDE534(),ISDE535()}
225       ),_or(
226         {ISDE542(),ISDE543()}
227       ),_or(
228         {ISDE550(),ISDE551()}
229       ),_or(
230         {ISDE558(),ISDE559()}
231       ),_or(
232         {ISDE566(),ISDE567()}
233       ),_or(
234         {ISDE574(),ISDE575()}
235       ),_or(
```

```
236         {ISDE582(),ISDE583()}
237       )}
238     ),1
239 )
240
241 --[[      RULE 27:     --]]
242 _between(
243   _sum(
244     {0,_or(
245       {ISDE025(),ISDE026()}
246     ),_or(
247       {ISDE033(),ISDE034()}
248     ),_or(
249       {ISDE048(),ISDE049()}
250     ),_or(
251       {ISDE056(),ISDE057()}
252     ),_or(
253       {ISDE064(),ISDE065()}
254     ),_or(
255       {ISDE072(),ISDE073()}
256     ),_or(
257       {ISDE080(),ISDE081()}
258     ),_or(
259       {ISDE088(),ISDE089()}
260     ),_or(
261       {ISDE096(),ISDE097()}
262     ),_or(
263       {ISDE104(),ISDE105()}
264     ),_or(
265       {ISDE112(),ISDE113()}
266     ),_or(
267       {ISDE120(),ISDE121()}
268     ),_or(
269       {ISDE128(),ISDE129()}
270     ),_or(
271       {ISDE136(),ISDE137()}
272     ),_or(
273       {ISDE144(),ISDE145()}
274     ),_or(
275       {ISDE152(),ISDE153()}
276     ),_or(
277       {ISDE160(),ISDE161()}
278     ),_or(
279       {ISDE168(),ISDE169()}
280     ),_or(
281       {ISDE176(),ISDE177()}
282     ),_or(
283       {ISDE184(),ISDE185()}
284     ),_or(
285       {ISDE192(),ISDE193()}
286     ),_or(
287       {ISDE200(),ISDE201()}
288     ),_or(
289       {ISDE208(),ISDE209()}
290     ),_or(
291       {ISDE216(),ISDE217()}
```

```
292     ),_or(
293       {ISDE224(),ISDE225()}
294     ),_or(
295       {ISDE232(),ISDE233()}
296     ),_or(
297       {ISDE240(),ISDE241()}
298     ),_or(
299       {ISDE248(),ISDE249()}
300     ),_or(
301       {ISDE256(),ISDE257()}
302     ),_or(
303       {ISDE264(),ISDE265()}
304     ),_or(
305       {ISDE272(),ISDE273()}
306     ),_or(
307       {ISDE280(),ISDE281()}
308     ),_or(
309       {ISDE288(),ISDE289()}
310     ),_or(
311       {ISDE296(),ISDE297()}
312     ),_or(
313       {ISDE304(),ISDE305()}
314     ),_or(
315       {ISDE312(),ISDE313()}
316     ),_or(
317       {ISDE320(),ISDE321()}
318     ),_or(
319       {ISDE328(),ISDE329()}
320     ),_or(
321       {ISDE336(),ISDE337()}
322     ),_or(
323       {ISDE344(),ISDE345()}
324     ),_or(
325       {ISDE366(),ISDE367()}
326     ),_or(
327       {ISDE374(),ISDE375()}
328     ),_or(
329       {ISDE382(),ISDE383()}
330     ),_or(
331       {ISDE390(),ISDE391()}
332     ),_or(
333       {ISDE398(),ISDE399()}
334     ),_or(
335       {ISDE406(),ISDE407()}
336     ),_or(
337       {ISDE414(),ISDE415()}
338     ),_or(
339       {ISDE422(),ISDE423()}
340     ),_or(
341       {ISDE430(),ISDE431()}
342     ),_or(
343       {ISDE438(),ISDE439()}
344     ),_or(
345       {ISDE446(),ISDE447()}
346     ),_or(
347       {ISDE454(),ISDE455()}
```

```
348       ),_or(
349         {ISDE462(),ISDE463()}
350       ),_or(
351         {ISDE470(),ISDE471()}
352       ),_or(
353         {ISDE478(),ISDE479()}
354       ),_or(
355         {ISDE486(),ISDE487()}
356       ),_or(
357         {ISDE494(),ISDE495()}
358       ),_or(
359         {ISDE502(),ISDE503()}
360       ),_or(
361         {ISDE510(),ISDE511()}
362       ),_or(
363         {ISDE518(),ISDE519()}
364       ),_or(
365         {ISDE526(),ISDE527()}
366       ),_or(
367         {ISDE534(),ISDE535()}
368       ),_or(
369         {ISDE542(),ISDE543()}
370       ),_or(
371         {ISDE550(),ISDE551()}
372       ),_or(
373         {ISDE558(),ISDE559()}
374       ),_or(
375         {ISDE566(),ISDE567()}
376       ),_or(
377         {ISDE574(),ISDE575()}
378       ),_or(
379         {ISDE582(),ISDE583()}
380       )}
381     ),2,5,1,1
382   )
383
384   --[[      RULE 28:      --]]
385   _ge(
386     _sum(
387       {0,_or(
388         {ISDE025(),ISDE026()}
389       ),_or(
390         {ISDE033(),ISDE034()}
391       ),_or(
392         {ISDE048(),ISDE049()}
393       ),_or(
394         {ISDE056(),ISDE057()}
395       ),_or(
396         {ISDE064(),ISDE065()}
397       ),_or(
398         {ISDE072(),ISDE073()}
399       ),_or(
400         {ISDE080(),ISDE081()}
401       ),_or(
402         {ISDE088(),ISDE089()}
403       ),_or(
```

```
404         {ISDE096(),ISDE097()}
405       ),_or(
406         {ISDE104(),ISDE105()}
407       ),_or(
408         {ISDE112(),ISDE113()}
409       ),_or(
410         {ISDE120(),ISDE121()}
411       ),_or(
412         {ISDE128(),ISDE129()}
413       ),_or(
414         {ISDE136(),ISDE137()}
415       ),_or(
416         {ISDE144(),ISDE145()}
417       ),_or(
418         {ISDE152(),ISDE153()}
419       ),_or(
420         {ISDE160(),ISDE161()}
421       ),_or(
422         {ISDE168(),ISDE169()}
423       ),_or(
424         {ISDE176(),ISDE177()}
425       ),_or(
426         {ISDE184(),ISDE185()}
427       ),_or(
428         {ISDE192(),ISDE193()}
429       ),_or(
430         {ISDE200(),ISDE201()}
431       ),_or(
432         {ISDE208(),ISDE209()}
433       ),_or(
434         {ISDE216(),ISDE217()}
435       ),_or(
436         {ISDE224(),ISDE225()}
437       ),_or(
438         {ISDE232(),ISDE233()}
439       ),_or(
440         {ISDE240(),ISDE241()}
441       ),_or(
442         {ISDE248(),ISDE249()}
443       ),_or(
444         {ISDE256(),ISDE257()}
445       ),_or(
446         {ISDE264(),ISDE265()}
447       ),_or(
448         {ISDE272(),ISDE273()}
449       ),_or(
450         {ISDE280(),ISDE281()}
451       ),_or(
452         {ISDE288(),ISDE289()}
453       ),_or(
454         {ISDE296(),ISDE297()}
455       ),_or(
456         {ISDE304(),ISDE305()}
457       ),_or(
458         {ISDE312(),ISDE313()}
459       ),_or(
```

```
460         {ISDE320(),ISDE321()}
461       ),_or(
462         {ISDE328(),ISDE329()}
463       ),_or(
464         {ISDE336(),ISDE337()}
465       ),_or(
466         {ISDE344(),ISDE345()}
467       ),_or(
468         {ISDE366(),ISDE367()}
469       ),_or(
470         {ISDE374(),ISDE375()}
471       ),_or(
472         {ISDE382(),ISDE383()}
473       ),_or(
474         {ISDE390(),ISDE391()}
475       ),_or(
476         {ISDE398(),ISDE399()}
477       ),_or(
478         {ISDE406(),ISDE407()}
479       ),_or(
480         {ISDE414(),ISDE415()}
481       ),_or(
482         {ISDE422(),ISDE423()}
483       ),_or(
484         {ISDE430(),ISDE431()}
485       ),_or(
486         {ISDE438(),ISDE439()}
487       ),_or(
488         {ISDE446(),ISDE447()}
489       ),_or(
490         {ISDE454(),ISDE455()}
491       ),_or(
492         {ISDE462(),ISDE463()}
493       ),_or(
494         {ISDE470(),ISDE471()}
495       ),_or(
496         {ISDE478(),ISDE479()}
497       ),_or(
498         {ISDE486(),ISDE487()}
499       ),_or(
500         {ISDE494(),ISDE495()}
501       ),_or(
502         {ISDE502(),ISDE503()}
503       ),_or(
504         {ISDE510(),ISDE511()}
505       ),_or(
506         {ISDE518(),ISDE519()}
507       ),_or(
508         {ISDE526(),ISDE527()}
509       ),_or(
510         {ISDE534(),ISDE535()}
511       ),_or(
512         {ISDE542(),ISDE543()}
513       ),_or(
514         {ISDE550(),ISDE551()}
515       ),_or(
```

```
516         {ISDE558(),ISDE559()}
517       ),_or(
518         {ISDE566(),ISDE567()}
519       ),_or(
520         {ISDE574(),ISDE575()}
521       ),_or(
522         {ISDE582(),ISDE583()}
523       )}
524     ),6
525   )
526
527   --[[      RULE 29:     --]]
528   _ge(
529     _sum(
530       {0,ISDE027(),ISDE035(),ISDE050(),ISDE058(),ISDE066(),ISDE074(),ISDE082(),ISDE090(),ISDE098(),
          ISDE106(),ISDE114(),ISDE122(),ISDE130(),ISDE138(),ISDE146(),ISDE154(),ISDE162(),ISDE170(),
          ISDE178(),ISDE186(),ISDE194(),ISDE202(),ISDE210(),ISDE218(),ISDE226(),ISDE234(),ISDE242(),
          ISDE250(),ISDE258(),ISDE266(),ISDE274(),ISDE282(),ISDE290(),ISDE298(),ISDE306(),ISDE314(),
          ISDE322(),ISDE330(),ISDE338(),ISDE346(),ISDE368(),ISDE376(),ISDE384(),ISDE392(),ISDE400(),
          ISDE408(),ISDE416(),ISDE424(),ISDE432(),ISDE440(),ISDE448(),ISDE456(),ISDE464(),ISDE472(),
          ISDE480(),ISDE488(),ISDE496(),ISDE504(),ISDE512(),ISDE520(),ISDE528(),ISDE536(),ISDE544(),
          ISDE552(),ISDE560(),ISDE568(),ISDE576(),ISDE584()}
531     ),1
532   )
533
534   --[[      RULE 30:     --]]
535   _ge(
536     _sum(
537       {0,ISDE028(),ISDE036(),ISDE051(),ISDE059(),ISDE067(),ISDE075(),ISDE083(),ISDE091(),ISDE099(),
          ISDE107(),ISDE115(),ISDE123(),ISDE131(),ISDE139(),ISDE147(),ISDE155(),ISDE163(),ISDE171(),
          ISDE179(),ISDE187(),ISDE195(),ISDE203(),ISDE211(),ISDE219(),ISDE227(),ISDE235(),ISDE243(),
          ISDE251(),ISDE259(),ISDE267(),ISDE275(),ISDE283(),ISDE291(),ISDE299(),ISDE307(),ISDE315(),
          ISDE323(),ISDE331(),ISDE339(),ISDE347(),ISDE369(),ISDE377(),ISDE385(),ISDE393(),ISDE401(),
          ISDE409(),ISDE417(),ISDE425(),ISDE433(),ISDE441(),ISDE449(),ISDE457(),ISDE465(),ISDE473(),
          ISDE481(),ISDE489(),ISDE497(),ISDE505(),ISDE513(),ISDE521(),ISDE529(),ISDE537(),ISDE545(),
          ISDE553(),ISDE561(),ISDE569(),ISDE577(),ISDE585()}
538     ),1
539   )
540
541   --[[      RULE 31:     --]]
542   _ge(
543     _sum(
544       {0,_and(
545         {_or(
546           {ISDE285(),ISDE286()}
547         ),_or(
548           {ISDE288(),ISDE289()}
549         )}
550       ),_and(
551         {_or(
552           {ISDE293(),ISDE294()}
553         ),_or(
554           {ISDE296(),ISDE297()}
555         )}
556       ),_and(
557         {_or(
```

```
558         {ISDE301(),ISDE302()}
559       ),_or(
560         {ISDE304(),ISDE305()}
561       )}
562     ),_and(
563       {_or(
564         {ISDE309(),ISDE310()}
565       ),_or(
566         {ISDE312(),ISDE313()}
567       )}
568     ),_and(
569       {_or(
570         {ISDE317(),ISDE318()}
571       ),_or(
572         {ISDE320(),ISDE321()}
573       )}
574     ),_and(
575       {_or(
576         {ISDE325(),ISDE326()}
577       ),_or(
578         {ISDE328(),ISDE329()}
579       )}
580     ),_and(
581       {_or(
582         {ISDE333(),ISDE334()}
583       ),_or(
584         {ISDE336(),ISDE337()}
585       )}
586     ),_and(
587       {_or(
588         {ISDE341(),ISDE342()}
589       ),_or(
590         {ISDE344(),ISDE345()}
591       )}
592     ),_and(
593       {_or(
594         {ISDE507(),ISDE508()}
595       ),_or(
596         {ISDE510(),ISDE511()}
597       )}
598     ),_and(
599       {_or(
600         {ISDE515(),ISDE516()}
601       ),_or(
602         {ISDE518(),ISDE519()}
603       )}
604     ),_and(
605       {_or(
606         {ISDE523(),ISDE524()}
607       ),_or(
608         {ISDE526(),ISDE527()}
609       )}
610     ),_and(
611       {_or(
612         {ISDE531(),ISDE532()}
613       ),_or(
```

```
614           {ISDE534(),ISDE535()}
615         )}
616       ),_and(
617         {_or(
618           {ISDE539(),ISDE540()}
619         ),_or(
620           {ISDE542(),ISDE543()}
621         )}
622       ),_and(
623         {_or(
624           {ISDE547(),ISDE548()}
625         ),_or(
626           {ISDE550(),ISDE551()}
627         )}
628       ),_and(
629         {_or(
630           {ISDE555(),ISDE556()}
631         ),_or(
632           {ISDE558(),ISDE559()}
633         )}
634       ),_and(
635         {_or(
636           {ISDE563(),ISDE564()}
637         ),_or(
638           {ISDE566(),ISDE567()}
639         )}
640       ),_and(
641         {_or(
642           {ISDE571(),ISDE572()}
643         ),_or(
644           {ISDE574(),ISDE575()}
645         )}
646       ),_and(
647         {_or(
648           {ISDE579(),ISDE580()}
649         ),_or(
650           {ISDE582(),ISDE583()}
651         )}
652       )}
653     ),1
654 )
655
656 --[[      RULE 32:      --]]
657 _ge(
658   _sum(
659     {0,_and(
660       {ISDE284(),_or(
661         {ISDE288(),ISDE289()}
662       )}
663     ),_and(
664       {ISDE292(),_or(
665         {ISDE296(),ISDE297()}
666       )}
667     ),_and(
668       {ISDE300(),_or(
669         {ISDE304(),ISDE305()}
```

```
670         )}
671       ),_and(
672         {ISDE308(),_or(
673           {ISDE312(),ISDE313()}
674         )}
675       ),_and(
676         {ISDE316(),_or(
677           {ISDE320(),ISDE321()}
678         )}
679       ),_and(
680         {ISDE324(),_or(
681           {ISDE328(),ISDE329()}
682         )}
683       ),_and(
684         {ISDE332(),_or(
685           {ISDE336(),ISDE337()}
686         )}
687       ),_and(
688         {ISDE340(),_or(
689           {ISDE344(),ISDE345()}
690         )}
691       ),_and(
692         {ISDE506(),_or(
693           {ISDE510(),ISDE511()}
694         )}
695       ),_and(
696         {ISDE514(),_or(
697           {ISDE518(),ISDE519()}
698         )}
699       ),_and(
700         {ISDE522(),_or(
701           {ISDE526(),ISDE527()}
702         )}
703       ),_and(
704         {ISDE530(),_or(
705           {ISDE534(),ISDE535()}
706         )}
707       ),_and(
708         {ISDE538(),_or(
709           {ISDE542(),ISDE543()}
710         )}
711       ),_and(
712         {ISDE546(),_or(
713           {ISDE550(),ISDE551()}
714         )}
715       ),_and(
716         {ISDE554(),_or(
717           {ISDE558(),ISDE559()}
718         )}
719       ),_and(
720         {ISDE562(),_or(
721           {ISDE566(),ISDE567()}
722         )}
723       ),_and(
724         {ISDE570(),_or(
725           {ISDE574(),ISDE575()}
```

```
726         )}
727       ),_and(
728         {ISDE578(),_or(
729           {ISDE582(),ISDE583()}
730         )}
731       )}
732     ),1
733   )
734
735   --[[      RULE 33:     --]]
736   _or(
737     {_eq(
738       _sum(
739         {0,_and(
740           {_eq(
741             _sum(
742               {0,_ge(
743                 _sum(
744                   {0,_ge(
745                     _sum(
746                       {0,ISDE108(),ISDE109(),ISDE110(),ISDE111(),ISDE112(),ISDE113(),ISDE114(),ISDE115()}
747                     ),1
748                   ),_ge(
749                     _sum(
750                       {0,ISDE124(),ISDE125(),ISDE126(),ISDE127(),ISDE128(),ISDE129(),ISDE130(),ISDE131()}
751                     ),1
752                   ),_ge(
753                     _sum(
754                       {0,ISDE204(),ISDE205(),ISDE206(),ISDE207(),ISDE208(),ISDE209(),ISDE210(),ISDE211()}
755                     ),1
756                   )}
757                 ),3
758               ),_ge(
759                 _sum(
760                   {0,_ge(
761                     _sum(
762                       {0,ISDE108(),ISDE109(),ISDE110(),ISDE111(),ISDE112(),ISDE113(),ISDE114(),ISDE115()}
763                     ),1
764                   ),_ge(
765                     _sum(
766                       {0,ISDE140(),ISDE141(),ISDE142(),ISDE143(),ISDE144(),ISDE145(),ISDE146(),ISDE147()}
767                     ),1
768                   ),_ge(
769                     _sum(
770                       {0,ISDE220(),ISDE221(),ISDE222(),ISDE223(),ISDE224(),ISDE225(),ISDE226(),ISDE227()}
771                     ),1
772                   ),_ge(
773                     _sum(
774                       {0,ISDE284(),ISDE285(),ISDE286(),ISDE287(),ISDE288(),ISDE289(),ISDE290(),ISDE291()}
775                     ),1
776                   )}
777                 ),3
778               ),_ge(
779                 _sum(
780                   {0,_ge(
781                     _sum(
```

```
782                 {0,ISDE108(),ISDE109(),ISDE110(),ISDE111(),ISDE112(),ISDE113(),ISDE114(),ISDE115()}
783               ),1
784             ),_ge(
785               _sum(
786                 {0,ISDE156(),ISDE157(),ISDE158(),ISDE159(),ISDE160(),ISDE161(),ISDE162(),ISDE163()}
787               ),1
788             ),_ge(
789               _sum(
790                 {0,ISDE236(),ISDE237(),ISDE238(),ISDE239(),ISDE240(),ISDE241(),ISDE242(),ISDE243()}
791               ),1
792             ),_ge(
793               _sum(
794                 {0,ISDE300(),ISDE301(),ISDE302(),ISDE303(),ISDE304(),ISDE305(),ISDE306(),ISDE307()}
795               ),1
796             )}
797           ),3
798         ),_ge(
799           _sum(
800             {0,_ge(
801               _sum(
802                 {0,ISDE108(),ISDE109(),ISDE110(),ISDE111(),ISDE112(),ISDE113(),ISDE114(),ISDE115()}
803               ),1
804             ),_ge(
805               _sum(
806                 {0,ISDE172(),ISDE173(),ISDE174(),ISDE175(),ISDE176(),ISDE177(),ISDE178(),ISDE179()}
807               ),1
808             ),_ge(
809               _sum(
810                 {0,ISDE252(),ISDE253(),ISDE254(),ISDE255(),ISDE256(),ISDE257(),ISDE258(),ISDE259()}
811               ),1
812             ),_ge(
813               _sum(
814                 {0,ISDE316(),ISDE317(),ISDE318(),ISDE319(),ISDE320(),ISDE321(),ISDE322(),ISDE323()}
815               ),1
816             )}
817           ),3
818         ),_ge(
819           _sum(
820             {0,_ge(
821               _sum(
822                 {0,ISDE108(),ISDE109(),ISDE110(),ISDE111(),ISDE112(),ISDE113(),ISDE114(),ISDE115()}
823               ),1
824             ),_ge(
825               _sum(
826                 {0,ISDE188(),ISDE189(),ISDE190(),ISDE191(),ISDE192(),ISDE193(),ISDE194(),ISDE195()}
827               ),1
828             ),_ge(
829               _sum(
830                 {0,ISDE268(),ISDE269(),ISDE270(),ISDE271(),ISDE272(),ISDE273(),ISDE274(),ISDE275()}
831               ),1
832             ),_ge(
833               _sum(
834                 {0,ISDE332(),ISDE333(),ISDE334(),ISDE335(),ISDE336(),ISDE337(),ISDE338(),ISDE339()}
835               ),1
836             )}
837           ),3
```

```
838             )}
839           ),1
840         ),_neg(
841           _ge(
842             _sum(
843               {0,_ge(
844                 _sum(
845                   {0,_ge(
846                     _sum(
847                       {0,ISDE108(),ISDE109(),ISDE110(),ISDE111(),ISDE112(),ISDE113(),ISDE114(),ISDE115()
                          }
848                     ),1
849                   ),_ge(
850                     _sum(
851                       {0,ISDE124(),ISDE125(),ISDE126(),ISDE127(),ISDE128(),ISDE129(),ISDE130(),ISDE131()
                          }
852                     ),1
853                   ),_ge(
854                     _sum(
855                       {0,ISDE204(),ISDE205(),ISDE206(),ISDE207(),ISDE208(),ISDE209(),ISDE210(),ISDE211()
                          }
856                     ),1
857                   )}
858                 ),2
859               ),_ge(
860                 _sum(
861                   {0,_ge(
862                     _sum(
863                       {0,ISDE108(),ISDE109(),ISDE110(),ISDE111(),ISDE112(),ISDE113(),ISDE114(),ISDE115()
                          }
864                     ),1
865                   ),_ge(
866                     _sum(
867                       {0,ISDE140(),ISDE141(),ISDE142(),ISDE143(),ISDE144(),ISDE145(),ISDE146(),ISDE147()
                          }
868                     ),1
869                   ),_ge(
870                     _sum(
871                       {0,ISDE220(),ISDE221(),ISDE222(),ISDE223(),ISDE224(),ISDE225(),ISDE226(),ISDE227()
                          }
872                     ),1
873                   ),_ge(
874                     _sum(
875                       {0,ISDE284(),ISDE285(),ISDE286(),ISDE287(),ISDE288(),ISDE289(),ISDE290(),ISDE291()
                          }
876                     ),1
877                   )}
878                 ),2
879               ),_ge(
880                 _sum(
881                   {0,_ge(
882                     _sum(
883                       {0,ISDE108(),ISDE109(),ISDE110(),ISDE111(),ISDE112(),ISDE113(),ISDE114(),ISDE115()
                          }
884                     ),1
885                   ),_ge(
```

```
886                 _sum(
887                   {0,ISDE156(),ISDE157(),ISDE158(),ISDE159(),ISDE160(),ISDE161(),ISDE162(),ISDE163()
                        }
888                 ),1
889               ),_ge(
890                 _sum(
891                   {0,ISDE236(),ISDE237(),ISDE238(),ISDE239(),ISDE240(),ISDE241(),ISDE242(),ISDE243()
                        }
892                 ),1
893               ),_ge(
894                 _sum(
895                   {0,ISDE300(),ISDE301(),ISDE302(),ISDE303(),ISDE304(),ISDE305(),ISDE306(),ISDE307()
                        }
896                 ),1
897               )}
898             ),2
899           ),_ge(
900             _sum(
901               {0,_ge(
902                 _sum(
903                   {0,ISDE108(),ISDE109(),ISDE110(),ISDE111(),ISDE112(),ISDE113(),ISDE114(),ISDE115()
                        }
904                 ),1
905               ),_ge(
906                 _sum(
907                   {0,ISDE172(),ISDE173(),ISDE174(),ISDE175(),ISDE176(),ISDE177(),ISDE178(),ISDE179()
                        }
908                 ),1
909               ),_ge(
910                 _sum(
911                   {0,ISDE252(),ISDE253(),ISDE254(),ISDE255(),ISDE256(),ISDE257(),ISDE258(),ISDE259()
                        }
912                 ),1
913               ),_ge(
914                 _sum(
915                   {0,ISDE316(),ISDE317(),ISDE318(),ISDE319(),ISDE320(),ISDE321(),ISDE322(),ISDE323()
                        }
916                 ),1
917               )}
918             ),2
919           ),_ge(
920             _sum(
921               {0,_ge(
922                 _sum(
923                   {0,ISDE108(),ISDE109(),ISDE110(),ISDE111(),ISDE112(),ISDE113(),ISDE114(),ISDE115()
                        }
924                 ),1
925               ),_ge(
926                 _sum(
927                   {0,ISDE188(),ISDE189(),ISDE190(),ISDE191(),ISDE192(),ISDE193(),ISDE194(),ISDE195()
                        }
928                 ),1
929               ),_ge(
930                 _sum(
931                   {0,ISDE268(),ISDE269(),ISDE270(),ISDE271(),ISDE272(),ISDE273(),ISDE274(),ISDE275()
                        }
```

```
932                     ),1
933                   ),_ge(
934                     _sum(
935                       {0,ISDE332(),ISDE333(),ISDE334(),ISDE335(),ISDE336(),ISDE337(),ISDE338(),ISDE339()
                            }
936                     ),1
937                   )}
938                 ),2
939               )}
940             ),2
941           )
942         )}
943       ),_and(
944         {_eq(
945           _sum(
946             {0,_ge(
947               _sum(
948                 {0,_ge(
949                   _sum(
950                     {0,ISDE116(),ISDE117(),ISDE118(),ISDE119(),ISDE120(),ISDE121(),ISDE122(),ISDE123()}
951                   ),1
952                 ),_ge(
953                   _sum(
954                     {0,ISDE132(),ISDE133(),ISDE134(),ISDE135(),ISDE136(),ISDE137(),ISDE138(),ISDE139()}
955                   ),1
956                 ),_ge(
957                   _sum(
958                     {0,ISDE212(),ISDE213(),ISDE214(),ISDE215(),ISDE216(),ISDE217(),ISDE218(),ISDE219()}
959                   ),1
960                 )}
961               ),3
962             ),_ge(
963               _sum(
964                 {0,_ge(
965                   _sum(
966                     {0,ISDE116(),ISDE117(),ISDE118(),ISDE119(),ISDE120(),ISDE121(),ISDE122(),ISDE123()}
967                   ),1
968                 ),_ge(
969                   _sum(
970                     {0,ISDE148(),ISDE149(),ISDE150(),ISDE151(),ISDE152(),ISDE153(),ISDE154(),ISDE155()}
971                   ),1
972                 ),_ge(
973                   _sum(
974                     {0,ISDE228(),ISDE229(),ISDE230(),ISDE231(),ISDE232(),ISDE233(),ISDE234(),ISDE235()}
975                   ),1
976                 ),_ge(
977                   _sum(
978                     {0,ISDE292(),ISDE293(),ISDE294(),ISDE295(),ISDE296(),ISDE297(),ISDE298(),ISDE299()}
979                   ),1
980                 )}
981               ),3
982             ),_ge(
983               _sum(
984                 {0,_ge(
985                   _sum(
986                     {0,ISDE116(),ISDE117(),ISDE118(),ISDE119(),ISDE120(),ISDE121(),ISDE122(),ISDE123()}
```

```
987                 ),1
988               ),_ge(
989                 _sum(
990                   {0,ISDE164(),ISDE165(),ISDE166(),ISDE167(),ISDE168(),ISDE169(),ISDE170(),ISDE171()}
991                 ),1
992               ),_ge(
993                 _sum(
994                   {0,ISDE244(),ISDE245(),ISDE246(),ISDE247(),ISDE248(),ISDE249(),ISDE250(),ISDE251()}
995                 ),1
996               ),_ge(
997                 _sum(
998                   {0,ISDE308(),ISDE309(),ISDE310(),ISDE311(),ISDE312(),ISDE313(),ISDE314(),ISDE315()}
999                 ),1
1000              )}
1001            ),3
1002          ),_ge(
1003            _sum(
1004              {0,_ge(
1005                _sum(
1006                  {0,ISDE116(),ISDE117(),ISDE118(),ISDE119(),ISDE120(),ISDE121(),ISDE122(),ISDE123()}
1007                ),1
1008              ),_ge(
1009                _sum(
1010                  {0,ISDE180(),ISDE181(),ISDE182(),ISDE183(),ISDE184(),ISDE185(),ISDE186(),ISDE187()}
1011                ),1
1012              ),_ge(
1013                _sum(
1014                  {0,ISDE260(),ISDE261(),ISDE262(),ISDE263(),ISDE264(),ISDE265(),ISDE266(),ISDE267()}
1015                ),1
1016              ),_ge(
1017                _sum(
1018                  {0,ISDE324(),ISDE325(),ISDE326(),ISDE327(),ISDE328(),ISDE329(),ISDE330(),ISDE331()}
1019                ),1
1020              )}
1021            ),3
1022          ),_ge(
1023            _sum(
1024              {0,_ge(
1025                _sum(
1026                  {0,ISDE116(),ISDE117(),ISDE118(),ISDE119(),ISDE120(),ISDE121(),ISDE122(),ISDE123()}
1027                ),1
1028              ),_ge(
1029                _sum(
1030                  {0,ISDE196(),ISDE197(),ISDE198(),ISDE199(),ISDE200(),ISDE201(),ISDE202(),ISDE203()}
1031                ),1
1032              ),_ge(
1033                _sum(
1034                  {0,ISDE276(),ISDE277(),ISDE278(),ISDE279(),ISDE280(),ISDE281(),ISDE282(),ISDE283()}
1035                ),1
1036              ),_ge(
1037                _sum(
1038                  {0,ISDE340(),ISDE341(),ISDE342(),ISDE343(),ISDE344(),ISDE345(),ISDE346(),ISDE347()}
1039                ),1
1040              )}
1041            ),3
1042          )}
```

```
1043            ),1
1044          ),_neg(
1045            _ge(
1046              _sum(
1047                {0,_ge(
1048                  _sum(
1049                    {0,_ge(
1050                      _sum(
1051                        {0,ISDE116(),ISDE117(),ISDE118(),ISDE119(),ISDE120(),ISDE121(),ISDE122(),ISDE123()
                            }
1052                      ),1
1053                    ),_ge(
1054                      _sum(
1055                        {0,ISDE132(),ISDE133(),ISDE134(),ISDE135(),ISDE136(),ISDE137(),ISDE138(),ISDE139()
                            }
1056                      ),1
1057                    ),_ge(
1058                      _sum(
1059                        {0,ISDE212(),ISDE213(),ISDE214(),ISDE215(),ISDE216(),ISDE217(),ISDE218(),ISDE219()
                            }
1060                      ),1
1061                    )}
1062                  ),2
1063                ),_ge(
1064                  _sum(
1065                    {0,_ge(
1066                      _sum(
1067                        {0,ISDE116(),ISDE117(),ISDE118(),ISDE119(),ISDE120(),ISDE121(),ISDE122(),ISDE123()
                            }
1068                      ),1
1069                    ),_ge(
1070                      _sum(
1071                        {0,ISDE148(),ISDE149(),ISDE150(),ISDE151(),ISDE152(),ISDE153(),ISDE154(),ISDE155()
                            }
1072                      ),1
1073                    ),_ge(
1074                      _sum(
1075                        {0,ISDE228(),ISDE229(),ISDE230(),ISDE231(),ISDE232(),ISDE233(),ISDE234(),ISDE235()
                            }
1076                      ),1
1077                    ),_ge(
1078                      _sum(
1079                        {0,ISDE292(),ISDE293(),ISDE294(),ISDE295(),ISDE296(),ISDE297(),ISDE298(),ISDE299()
                            }
1080                      ),1
1081                    )}
1082                  ),2
1083                ),_ge(
1084                  _sum(
1085                    {0,_ge(
1086                      _sum(
1087                        {0,ISDE116(),ISDE117(),ISDE118(),ISDE119(),ISDE120(),ISDE121(),ISDE122(),ISDE123()
                            }
1088                      ),1
1089                    ),_ge(
1090                      _sum(
```

```
1091                    {0,ISDE164(),ISDE165(),ISDE166(),ISDE167(),ISDE168(),ISDE169(),ISDE170(),ISDE171()
                          }
1092                  ),1
1093                ),_ge(
1094                  _sum(
1095                    {0,ISDE244(),ISDE245(),ISDE246(),ISDE247(),ISDE248(),ISDE249(),ISDE250(),ISDE251()
                          }
1096                  ),1
1097                ),_ge(
1098                  _sum(
1099                    {0,ISDE308(),ISDE309(),ISDE310(),ISDE311(),ISDE312(),ISDE313(),ISDE314(),ISDE315()
                          }
1100                  ),1
1101                )}
1102              ),2
1103            ),_ge(
1104              _sum(
1105                {0,_ge(
1106                  _sum(
1107                    {0,ISDE116(),ISDE117(),ISDE118(),ISDE119(),ISDE120(),ISDE121(),ISDE122(),ISDE123()
                          }
1108                  ),1
1109                ),_ge(
1110                  _sum(
1111                    {0,ISDE180(),ISDE181(),ISDE182(),ISDE183(),ISDE184(),ISDE185(),ISDE186(),ISDE187()
                          }
1112                  ),1
1113                ),_ge(
1114                  _sum(
1115                    {0,ISDE260(),ISDE261(),ISDE262(),ISDE263(),ISDE264(),ISDE265(),ISDE266(),ISDE267()
                          }
1116                  ),1
1117                ),_ge(
1118                  _sum(
1119                    {0,ISDE324(),ISDE325(),ISDE326(),ISDE327(),ISDE328(),ISDE329(),ISDE330(),ISDE331()
                          }
1120                  ),1
1121                )}
1122              ),2
1123            ),_ge(
1124              _sum(
1125                {0,_ge(
1126                  _sum(
1127                    {0,ISDE116(),ISDE117(),ISDE118(),ISDE119(),ISDE120(),ISDE121(),ISDE122(),ISDE123()
                          }
1128                  ),1
1129                ),_ge(
1130                  _sum(
1131                    {0,ISDE196(),ISDE197(),ISDE198(),ISDE199(),ISDE200(),ISDE201(),ISDE202(),ISDE203()
                          }
1132                  ),1
1133                ),_ge(
1134                  _sum(
1135                    {0,ISDE276(),ISDE277(),ISDE278(),ISDE279(),ISDE280(),ISDE281(),ISDE282(),ISDE283()
                          }
1136                  ),1
```

```
1137                    ),_ge(
1138                      _sum(
1139                        {0,ISDE340(),ISDE341(),ISDE342(),ISDE343(),ISDE344(),ISDE345(),ISDE346(),ISDE347()
                            }
1140                      ),1
1141                    )}
1142                  ),2
1143                )}
1144              ),2
1145            )
1146          )}
1147        )}
1148      ),1
1149    ),_eq(
1150      _sum(
1151        {0,_and(
1152          {_eq(
1153            _sum(
1154              {0,_and(
1155                {_ge(
1156                  _sum(
1157                    {0,ISDE426(),ISDE427(),ISDE428(),ISDE429(),ISDE430(),ISDE431(),ISDE432(),ISDE433()}
1158                  ),1
1159                ),_ge(
1160                  _sum(
1161                    {0,ISDE506(),ISDE507(),ISDE508(),ISDE509(),ISDE510(),ISDE511(),ISDE512(),ISDE513()}
1162                  ),1
1163                )}
1164              ),_and(
1165                {_ge(
1166                  _sum(
1167                    {0,ISDE442(),ISDE443(),ISDE444(),ISDE445(),ISDE446(),ISDE447(),ISDE448(),ISDE449()}
1168                  ),1
1169                ),_ge(
1170                  _sum(
1171                    {0,ISDE522(),ISDE523(),ISDE524(),ISDE525(),ISDE526(),ISDE527(),ISDE528(),ISDE529()}
1172                  ),1
1173                )}
1174              ),_and(
1175                {_ge(
1176                  _sum(
1177                    {0,ISDE458(),ISDE459(),ISDE460(),ISDE461(),ISDE462(),ISDE463(),ISDE464(),ISDE465()}
1178                  ),1
1179                ),_ge(
1180                  _sum(
1181                    {0,ISDE538(),ISDE539(),ISDE540(),ISDE541(),ISDE542(),ISDE543(),ISDE544(),ISDE545()}
1182                  ),1
1183                )}
1184              ),_and(
1185                {_ge(
1186                  _sum(
1187                    {0,ISDE474(),ISDE475(),ISDE476(),ISDE477(),ISDE478(),ISDE479(),ISDE480(),ISDE481()}
1188                  ),1
1189                ),_ge(
1190                  _sum(
1191                    {0,ISDE554(),ISDE555(),ISDE556(),ISDE557(),ISDE558(),ISDE559(),ISDE560(),ISDE561()}
```

```
1192                ),1
1193              )}
1194            ),_and(
1195              {_ge(
1196                _sum(
1197                  {0,ISDE490(),ISDE491(),ISDE492(),ISDE493(),ISDE494(),ISDE495(),ISDE496(),ISDE497()}
1198                ),1
1199              ),_ge(
1200                _sum(
1201                  {0,ISDE570(),ISDE571(),ISDE572(),ISDE573(),ISDE574(),ISDE575(),ISDE576(),ISDE577()}
1202                ),1
1203              )}
1204            )}
1205          ),1
1206        ),_neg(
1207          _ge(
1208            _sum(
1209              {0,_or(
1210                {_ge(
1211                  _sum(
1212                    {0,ISDE426(),ISDE427(),ISDE428(),ISDE429(),ISDE430(),ISDE431(),ISDE432(),ISDE433()}
1213                  ),1
1214                ),_ge(
1215                  _sum(
1216                    {0,ISDE506(),ISDE507(),ISDE508(),ISDE509(),ISDE510(),ISDE511(),ISDE512(),ISDE513()}
1217                  ),1
1218                )}
1219              ),_or(
1220                {_ge(
1221                  _sum(
1222                    {0,ISDE442(),ISDE443(),ISDE444(),ISDE445(),ISDE446(),ISDE447(),ISDE448(),ISDE449()}
1223                  ),1
1224                ),_ge(
1225                  _sum(
1226                    {0,ISDE522(),ISDE523(),ISDE524(),ISDE525(),ISDE526(),ISDE527(),ISDE528(),ISDE529()}
1227                  ),1
1228                )}
1229              ),_or(
1230                {_ge(
1231                  _sum(
1232                    {0,ISDE458(),ISDE459(),ISDE460(),ISDE461(),ISDE462(),ISDE463(),ISDE464(),ISDE465()}
1233                  ),1
1234                ),_ge(
1235                  _sum(
1236                    {0,ISDE538(),ISDE539(),ISDE540(),ISDE541(),ISDE542(),ISDE543(),ISDE544(),ISDE545()}
1237                  ),1
1238                )}
1239              ),_or(
1240                {_ge(
1241                  _sum(
1242                    {0,ISDE474(),ISDE475(),ISDE476(),ISDE477(),ISDE478(),ISDE479(),ISDE480(),ISDE481()}
1243                  ),1
1244                ),_ge(
1245                  _sum(
1246                    {0,ISDE554(),ISDE555(),ISDE556(),ISDE557(),ISDE558(),ISDE559(),ISDE560(),ISDE561()}
1247                  ),1
```

```
1248                )}
1249              ),_or(
1250                {_ge(
1251                  _sum(
1252                    {0,ISDE490(),ISDE491(),ISDE492(),ISDE493(),ISDE494(),ISDE495(),ISDE496(),ISDE497()}
1253                  ),1
1254                ),_ge(
1255                  _sum(
1256                    {0,ISDE570(),ISDE571(),ISDE572(),ISDE573(),ISDE574(),ISDE575(),ISDE576(),ISDE577()}
1257                  ),1
1258                )}
1259              )}
1260            ),2
1261          )
1262        )}
1263      ),_and(
1264        {_eq(
1265          _sum(
1266            {0,_and(
1267              {_ge(
1268                _sum(
1269                  {0,ISDE434(),ISDE435(),ISDE436(),ISDE437(),ISDE438(),ISDE439(),ISDE440(),ISDE441()}
1270                ),1
1271              ),_ge(
1272                _sum(
1273                  {0,ISDE514(),ISDE515(),ISDE516(),ISDE517(),ISDE518(),ISDE519(),ISDE520(),ISDE521()}
1274                ),1
1275              )}
1276            ),_and(
1277              {_ge(
1278                _sum(
1279                  {0,ISDE450(),ISDE451(),ISDE452(),ISDE453(),ISDE454(),ISDE455(),ISDE456(),ISDE457()}
1280                ),1
1281              ),_ge(
1282                _sum(
1283                  {0,ISDE530(),ISDE531(),ISDE532(),ISDE533(),ISDE534(),ISDE535(),ISDE536(),ISDE537()}
1284                ),1
1285              )}
1286            ),_and(
1287              {_ge(
1288                _sum(
1289                  {0,ISDE466(),ISDE467(),ISDE468(),ISDE469(),ISDE470(),ISDE471(),ISDE472(),ISDE473()}
1290                ),1
1291              ),_ge(
1292                _sum(
1293                  {0,ISDE546(),ISDE547(),ISDE548(),ISDE549(),ISDE550(),ISDE551(),ISDE552(),ISDE553()}
1294                ),1
1295              )}
1296            ),_and(
1297              {_ge(
1298                _sum(
1299                  {0,ISDE482(),ISDE483(),ISDE484(),ISDE485(),ISDE486(),ISDE487(),ISDE488(),ISDE489()}
1300                ),1
1301              ),_ge(
1302                _sum(
1303                  {0,ISDE562(),ISDE563(),ISDE564(),ISDE565(),ISDE566(),ISDE567(),ISDE568(),ISDE569()}
```

```
1304                ),1
1305              )}
1306            ),_and(
1307              {_ge(
1308                _sum(
1309                  {0,ISDE498(),ISDE499(),ISDE500(),ISDE501(),ISDE502(),ISDE503(),ISDE504(),ISDE505()}
1310                ),1
1311              ),_ge(
1312                _sum(
1313                  {0,ISDE578(),ISDE579(),ISDE580(),ISDE581(),ISDE582(),ISDE583(),ISDE584(),ISDE585()}
1314                ),1
1315              )}
1316            )}
1317          ),1
1318        ),_neg(
1319          _ge(
1320            _sum(
1321              {0,_or(
1322                {_ge(
1323                  _sum(
1324                    {0,ISDE434(),ISDE435(),ISDE436(),ISDE437(),ISDE438(),ISDE439(),ISDE440(),ISDE441()}
1325                  ),1
1326                ),_ge(
1327                  _sum(
1328                    {0,ISDE514(),ISDE515(),ISDE516(),ISDE517(),ISDE518(),ISDE519(),ISDE520(),ISDE521()}
1329                  ),1
1330                )}
1331              ),_or(
1332                {_ge(
1333                  _sum(
1334                    {0,ISDE450(),ISDE451(),ISDE452(),ISDE453(),ISDE454(),ISDE455(),ISDE456(),ISDE457()}
1335                  ),1
1336                ),_ge(
1337                  _sum(
1338                    {0,ISDE530(),ISDE531(),ISDE532(),ISDE533(),ISDE534(),ISDE535(),ISDE536(),ISDE537()}
1339                  ),1
1340                )}
1341              ),_or(
1342                {_ge(
1343                  _sum(
1344                    {0,ISDE466(),ISDE467(),ISDE468(),ISDE469(),ISDE470(),ISDE471(),ISDE472(),ISDE473()}
1345                  ),1
1346                ),_ge(
1347                  _sum(
1348                    {0,ISDE546(),ISDE547(),ISDE548(),ISDE549(),ISDE550(),ISDE551(),ISDE552(),ISDE553()}
1349                  ),1
1350                )}
1351              ),_or(
1352                {_ge(
1353                  _sum(
1354                    {0,ISDE482(),ISDE483(),ISDE484(),ISDE485(),ISDE486(),ISDE487(),ISDE488(),ISDE489()}
1355                  ),1
1356                ),_ge(
1357                  _sum(
1358                    {0,ISDE562(),ISDE563(),ISDE564(),ISDE565(),ISDE566(),ISDE567(),ISDE568(),ISDE569()}
1359                  ),1
```

```
1360                  )}
1361                ),_or(
1362                  {_ge(
1363                    _sum(
1364                      {0,ISDE498(),ISDE499(),ISDE500(),ISDE501(),ISDE502(),ISDE503(),ISDE504(),ISDE505()}
1365                    ),1
1366                  ),_ge(
1367                    _sum(
1368                      {0,ISDE578(),ISDE579(),ISDE580(),ISDE581(),ISDE582(),ISDE583(),ISDE584(),ISDE585()}
1369                    ),1
1370                  )}
1371                )}
1372              ),2
1373            )
1374          )}
1375        )}
1376      ),1
1377    )}
1378  )
1379
1380  --[[      RULE 34:      --]]
1381  _ge(
1382    _sum(
1383      {0,_and(
1384        {_or(
1385          {ISDE077(),ISDE078()}
1386        ),_or(
1387          {ISDE080(),ISDE081()}
1388        )}
1389      ),_and(
1390        {_or(
1391          {ISDE093(),ISDE094()}
1392        ),_or(
1393          {ISDE096(),ISDE097()}
1394        )}
1395      ),_and(
1396        {_or(
1397          {ISDE109(),ISDE110()}
1398        ),_or(
1399          {ISDE112(),ISDE113()}
1400        )}
1401      ),_and(
1402        {_or(
1403          {ISDE363(),ISDE364()}
1404        ),_or(
1405          {ISDE366(),ISDE367()}
1406        )}
1407      ),_and(
1408        {_or(
1409          {ISDE379(),ISDE380()}
1410        ),_or(
1411          {ISDE382(),ISDE383()}
1412        )}
1413      ),_and(
1414        {_or(
1415          {ISDE395(),ISDE396()}
```

```
1416        ),_or(
1417          {ISDE398(),ISDE399()}
1418        )}
1419      ),_and(
1420        {_or(
1421          {ISDE411(),ISDE412()}
1422        ),_or(
1423          {ISDE414(),ISDE415()}
1424        )}
1425      ),_and(
1426        {_or(
1427          {ISDE085(),ISDE086()}
1428        ),_or(
1429          {ISDE088(),ISDE089()}
1430        )}
1431      ),_and(
1432        {_or(
1433          {ISDE101(),ISDE102()}
1434        ),_or(
1435          {ISDE104(),ISDE105()}
1436        )}
1437      ),_and(
1438        {_or(
1439          {ISDE117(),ISDE118()}
1440        ),_or(
1441          {ISDE120(),ISDE121()}
1442        )}
1443      ),_and(
1444        {_or(
1445          {ISDE371(),ISDE372()}
1446        ),_or(
1447          {ISDE374(),ISDE375()}
1448        )}
1449      ),_and(
1450        {_or(
1451          {ISDE387(),ISDE388()}
1452        ),_or(
1453          {ISDE390(),ISDE391()}
1454        )}
1455      ),_and(
1456        {_or(
1457          {ISDE403(),ISDE404()}
1458        ),_or(
1459          {ISDE406(),ISDE407()}
1460        )}
1461      ),_and(
1462        {_or(
1463          {ISDE419(),ISDE420()}
1464        ),_or(
1465          {ISDE422(),ISDE423()}
1466        )}
1467      )}
1468    ),1
1469  )
1470
1471  --[[      RULE 35:      --]]
```

```
1472 _between(
1473   _sum(
1474     {0,_and(
1475       {_or(
1476         {ISDE077(),ISDE078()}
1477       ),_or(
1478         {ISDE080(),ISDE081()}
1479       )}
1480     ),_and(
1481       {_or(
1482         {ISDE093(),ISDE094()}
1483       ),_or(
1484         {ISDE096(),ISDE097()}
1485       )}
1486     ),_and(
1487       {_or(
1488         {ISDE109(),ISDE110()}
1489       ),_or(
1490         {ISDE112(),ISDE113()}
1491       )}
1492     ),_and(
1493       {_or(
1494         {ISDE363(),ISDE364()}
1495       ),_or(
1496         {ISDE366(),ISDE367()}
1497       )}
1498     ),_and(
1499       {_or(
1500         {ISDE379(),ISDE380()}
1501       ),_or(
1502         {ISDE382(),ISDE383()}
1503       )}
1504     ),_and(
1505       {_or(
1506         {ISDE395(),ISDE396()}
1507       ),_or(
1508         {ISDE398(),ISDE399()}
1509       )}
1510     ),_and(
1511       {_or(
1512         {ISDE411(),ISDE412()}
1513       ),_or(
1514         {ISDE414(),ISDE415()}
1515       )}
1516     ),_and(
1517       {_or(
1518         {ISDE085(),ISDE086()}
1519       ),_or(
1520         {ISDE088(),ISDE089()}
1521       )}
1522     ),_and(
1523       {_or(
1524         {ISDE101(),ISDE102()}
1525       ),_or(
1526         {ISDE104(),ISDE105()}
1527       )}
```

```
1528      ),_and(
1529        {_or(
1530          {ISDE117(),ISDE118()}
1531        ),_or(
1532          {ISDE120(),ISDE121()}
1533        )}
1534      ),_and(
1535        {_or(
1536          {ISDE371(),ISDE372()}
1537        ),_or(
1538          {ISDE374(),ISDE375()}
1539        )}
1540      ),_and(
1541        {_or(
1542          {ISDE387(),ISDE388()}
1543        ),_or(
1544          {ISDE390(),ISDE391()}
1545        )}
1546      ),_and(
1547        {_or(
1548          {ISDE403(),ISDE404()}
1549        ),_or(
1550          {ISDE406(),ISDE407()}
1551        )}
1552      ),_and(
1553        {_or(
1554          {ISDE419(),ISDE420()}
1555        ),_or(
1556          {ISDE422(),ISDE423()}
1557        )}
1558      )}
1559    ),2,5,1,1
1560  )
1561
1562  --[[      RULE 36:      --]]
1563  _neg(
1564    _ge(
1565      _sum(
1566        {0,ISDE037(),ISDE038(),ISDE039(),ISDE040(),ISDE041(),ISDE042(),ISDE043(),ISDE348(),ISDE349(),
          ISDE350(),ISDE351(),ISDE352(),ISDE353(),ISDE354(),ISDE355(),ISDE356(),ISDE357(),ISDE358(),
          ISDE359(),ISDE360(),ISDE361()}
1567      ),1
1568    )
1569  )
1570
1571  --[[      RULE 37:      --]]
1572  _ge(
1573    _sum(
1574      {0,ISDE037(),ISDE348(),ISDE355()}
1575    ),1
1576  )
1577
1578  --[[      RULE 38:      --]]
1579  _ge(
1580    _sum(
1581      {0,ISDE038(),ISDE349(),ISDE356()}
```

```
1582    ),1
1583  )
1584
1585  --[[      RULE 39:     --]]
1586  _ge(
1587    _sum(
1588      {0,ISDE039(),ISDE350(),ISDE357()}
1589    ),1
1590  )
1591
1592  --[[      RULE 40:     --]]
1593  _ge(
1594    _sum(
1595      {0,ISDE040(),ISDE351(),ISDE358()}
1596    ),1
1597  )
1598
1599  --[[      RULE 41:     --]]
1600  _ge(
1601    _sum(
1602      {0,ISDE041(),ISDE352(),ISDE359()}
1603    ),1
1604  )
1605
1606  --[[      RULE 42:     --]]
1607  _ge(
1608    _sum(
1609      {0,ISDE042(),ISDE353(),ISDE360()}
1610    ),1
1611  )
1612
1613  --[[      RULE 43:     --]]
1614  _ge(
1615    _sum(
1616      {0,ISDE043(),ISDE354(),ISDE361()}
1617    ),1
1618  )
1619
1620  --[[      RULE 44:     --]]
1621  ISDE586()
1622
1623  --[[      RULE 45:     --]]
1624  ISDE587()
1625
1626  --[[      RULE 46:     --]]
1627  _or(
1628    {ISDE588(),ISDE589()}
1629  )
1630
1631  --[[      RULE 47:     --]]
1632  _or(
1633    {ISDE590(),ISDE591()}
1634  )
1635
1636  --[[      RULE 48:     --]]
1637  ISDE592()
```

```
1638
1639 --[[    RULE 49:    --]]
1640 _ge(
1641   _sum(
1642     {0,ISDE593(),ISDE594(),ISDE595(),ISDE596()}
1643   ),1
1644 )
1645
1646 --[[    RULE 50:    --]]
1647 _and(
1648   {ISDE597(),_and(
1649     {ISDE622(),ISDE624()}
1650   )}
1651 )
1652
1653 --[[    RULE 51:    --]]
1654 _and(
1655   {ISDE599(),_and(
1656     {ISDE622(),ISDE625()}
1657   )}
1658 )
1659
1660 --[[    RULE 52:    --]]
1661 _and(
1662   {ISDE600(),_and(
1663     {ISDE622(),ISDE626()}
1664   )}
1665 )
1666
1667 --[[    RULE 53:    --]]
1668 _and(
1669   {ISDE598(),_and(
1670     {ISDE623(),ISDE624()}
1671   )}
1672 )
1673
1674 --[[    RULE 54:    --]]
1675 _and(
1676   {ISDE599(),_and(
1677     {ISDE623(),ISDE625()}
1678   )}
1679 )
1680
1681 --[[    RULE 55:    --]]
1682 _and(
1683   {ISDE600(),_and(
1684     {ISDE623(),ISDE626()}
1685   )}
1686 )
1687
1688 --[[    RULE 56:    --]]
1689 ISDE601()
1690
1691 --[[    RULE 57:    --]]
1692 ISDE602()
1693
```

```
1694 --[[      RULE 58:    --]]
1695 ISDE603()
1696
1697 --[[      RULE 59:    --]]
1698 ISDE604()
1699
1700 --[[      RULE 60:    --]]
1701 ISDE605()
1702
1703 --[[      RULE 61:    --]]
1704 ISDE606()
1705
1706 --[[      RULE 62:    --]]
1707 ISDE607()
1708
1709 --[[      RULE 63:    --]]
1710 ISDE608()
1711
1712 --[[      RULE 64:    --]]
1713 ISDE609()
1714
1715 --[[      RULE 65:    --]]
1716 ISDE610()
1717
1718 --[[      RULE 66:    --]]
1719 ISDE611()
1720
1721 --[[      RULE 67:    --]]
1722 ISDE612()
1723
1724 --[[      RULE 68:    --]]
1725 ISDE613()
1726
1727 --[[      RULE 69:    --]]
1728 ISDE614()
1729
1730 --[[      RULE 70:    --]]
1731 ISDE615()
1732
1733 --[[      RULE 71:    --]]
1734 ISDE616()
1735
1736 --[[      RULE 72:    --]]
1737 ISDE617()
1738
1739 --[[      RULE 73:    --]]
1740 ISDE618()
1741
1742 --[[      RULE 74:    --]]
1743 ISDE619()
1744
1745 --[[      RULE 75:    --]]
1746 ISDE620()
1747
1748 --[[      RULE 76:    --]]
1749 ISDE621()
```

Literaturverzeichnis

Bücher

1. **Janice S. Aikins, John Kunz, Edward H. Shortliffe, Robert Fallat**,
   PUFF: An Expert System for Interpretation of Pulmonary Function Data, Departments of Medicine and Computer Science, Stanford University, 1982;

2. **John W. Backus et al.**,
   Revised Report on the Algorithmic Language Algol 60, Journal of the British Computer Soc., London, 1963;

3. **Bruce Buchanan, Edward H. Shortliffe**,
   Rule-based Expert Systems, Addison-Wesley Publishing, Massachusetts, 1985;

4. **Charles F. Goldfarb**,
   The SGML Handbook, Clarendon Pr., Oxford, 1990;

5. **Roberto Ierusalimschy**,
   Programming in Lua, 2nd edition, Lua.org, Rio de Janeiro, 2006;

6. **Jörg Kindermann**,
   Experten Parsing. Parsing und Wissensrepräsentation im textheoretischen Rahmen, Buske Helmut Verlag GmbH, Hamburg, 1998;

7. **Stephen C. Kleene**,
   Mathematical Logic, John Wiley & Sons Inc, New York, 1967;

8. **Jan Łukasiewicz, L. Borkowski**,
   Selected Works, North-Holland Publ. Comp., Amsterdam, Warsaw, 1970;

9. **Jan Łukasiewicz, Fred Seddon**,
   Aristotle & Łukasiewicz on the Principle of Contradiction, Modern Logic Pub., Ames, 1996;

10. **William J. van Melle**,
System Aids in Constructing Consultation Programs, UMI Research Press, Ann Arbor, MI, 1981;

11. **Albert Menne**,
Einführung in die formale Logik, Wissenschaftliche Buchgesellschaft, [Abt. Verl.], Darmstadt, 1985;

12. **Manfred Schnabel**,
Expertensysteme in der Medizin - Eine Einführung mit Beispielen, Gustav Fischer, Stuttgart, 1996;

13. **John R. Searle**,
Mystery of Consciousness, Granta Books, London, 1997;

14. **Rudolf Seising**,
Die Fuzzifizierung der Systeme, Franz Steiner Verlag, Stuttgart, 2005;

15. **Edward H. Shortliffe**,
Computer-Based Medical Consultations: MYCIN, American Elsevier Publishing Company, New York, 1976;

16. **Edward H. Shortliffe, Bruce Buchanan**,
A Model of Inexact Reasoning in Medicine, American Elsevier Publishing Company, Massachusetts, 1984;

17. **Donald A. Waterman, Douglas Lenat, Frederick Hayes-Roth**,
Building Expert Systems, Addison-Wesley Publishing, Sydney, 1983;

18. **Reinhard Wilhelm, Dieter Mauer**,
Übersetzerbau. Theorie, Konstruktion, Generierung. 2. Auflage, Springer-Verlag, Berlin, 1997;

19. **Reinhard Wilhelm, Helmut Seidl**,
Übersetzerbau. Virtuelle Maschinen, Springer-Verlag, Berlin, 2007;

Journalartikel und Tagungsbände

20. **Klaus-Peter Adlassnig**,
A Fuzzy Logical Model of Computer-Assisted Medical Diagnosis, Methods of Information in Medicine, Nr. 19, Seiten 141–148 (1980);

21. **Klaus-Peter Adlassnig, Gernot Kolarz, Werner Scheithauer, Harald Effenberger, Georg Grabner**,
CADIAG: Approaches to Computer-Assisted Medical Diagnosis, Computers in Biology and Medicine, Nr. 15, Seiten 513–535 (1985);

22. **Klaus-Peter Adlassnig**,
Fuzzy Set Theory in Medical Diagnosis, IEEE Transactions on Systems, Man and Cybernetics, Nr. 16, Seiten 260–265 (1986);

23. **Karl Bögl, Klaus-Peter Adlassnig, Gernot Kolarz, Andreas Hatvan**,
RHEUMexpert: Medical Documentation and Diagnostic Decision Support in Rheumatology for the General Practitioner, OEGAI-Journal 18, Nr. 2, Seiten 5–6 (1999);

24. **Luiz de Figueiredo, Waldemar Celes, Roberto Ierusalimschy**,
Programming advanced control mechanisms with Lua coroutines, Game Programming Gems 6, Seiten 357–369, Charles River Media (2006);

25. **Sture Holm**,
A simple sequentially rejective multiple test procedure, Scandinavian Journal of Statistics, Nr. 6, Seiten 65–75 (1979);

26. **Donald E. Knuth**,
Backus Normal Form vs. Backus Naur Form, Communications of the ACM 7(12), Seiten 735–736 (1964);

27. **Gernot Kolarz, Klaus-Peter Adlassnig, Karl Bögl, Andreas Hatvan**,
RHEUMexpert: Ein Dokumentations- und Expertensystem für rheumatische Erkrankungen, Wiener MedizinischeWochenschrift 149, 19/20, Seiten 572–574 (1999);

28. **Gernot Kolarz, Franz Singer, Klaus-Peter Adlassnig, Karl Bögl, Andreas Hatvan**,
RHEUMexpert: Standardisierte Dokumentation rheumatischer Erkrankungen, In: Karl Bögl, Werner Horn, Klaus-Peter Adlassnig (Eds.) *Abstractband zum 2. Symposium „Medizinische Experten- und Wissensbasierte Systeme und Computergestützte OP-Navigation am AKH-Wien“*, Universität Wien, Wien, Österreich, 40 (1998);

29. **Peter J. F. Lucas**,
Certainty-factor-like structures in Bayesian belief networks, Knowledge-Based Systems, Vol. 14, Seiten 327–335 (2001);

30. **Jan Łukasiewicz**,
O logice trojwartosciowej, Ruch Filozoficny Nr. 5, Seiten 170–171 (1920); English transl. in [8];

31. **Rudolf Seising**,
From vagueness in medical thought to the foundations of fuzzy reasoning in medical diagnosis, Artificial Intelligence in Medicine, 38(3), Seiten 237–256 (2006);

32. **Steven Sweet**,
Think About It: Artificial Intelligence & Expert Systems, Systems & Processes, Vol. 3, Issue 4 (1999);

33. **Wolfgang Wahlster**,
KI-Verfahren zur Unterstützung der Ärztlichen Urteilsbildung, Brauer, W. (ed.): GI-11. Jahrestagung. (Informatik-Fachberichte Bd. 50), Berlin: Springer, Seiten 568–579 (2005);

34. **Sholom M. Weiss, Casimir A. Kulikowski, Aran Safir**,
Glaucoma Consultation by Computer, Computers in Biology and Medicine, Nr. 8, Seiten 25–40 (1978);

35. **Lotfi A. Zadeh**,
Fuzzy Sets, Information and Control, Nr. 8, Seiten 338–353 (1965).

Dissertationen und Diplomarbeiten

36. **Ramin Gharakhanzadeh**,
RheumaNet: Ein WWW-basiertes Konsultationssystem für die Rheumatologie. Diplomarbeit, Technische Universität Wien, Wien, 1999;

37. **Günter Kolousek**,
The System Architecture of an Integrated Medical Consultation system and its Implementation Based on Fuzzy Technology. Dissertation, Technische Universität Wien, Wien, 1997;

38. **Udo Kronberger**,
Wissenserwerbssystem für RHEUMexpert/Web. Magisterarbeit, Technische Universität Wien, Wien, 2006;

39. **Reinhard Pitsch**,
RHEUMexpert-II-Web. Magisterarbeit, Technische Universität Wien, Wien, 2005;

40. **Irfan Skiljan**,
RHEUMexpert-II: Ein Dokumentations- und differentialdiagnostisches Konsultationssystem für die Rheumatologie. Diplomarbeit, Technische Universität Wien, Wien, 2000;

Webreferenzen

41. **George Bebis**, Thresholding,
www.cs.unr.edu/~bebis/CS791E/Notes/Thresholding.pdf, 2006-04-25;

42. **Tim Berners-Lee**, Information Management: A Proposal,
www.w3.org/History/1989/proposal.html, 2007-08-08;

43. **Werner Horn**,
Kausale Modelle bei entscheidungsunterstützenden Systemen in der Medizin,
www.ofai.at/cgi-bin/tr-online?author+Horn, 2007-09-27;

44. **keplerproject.org**,
www.keplerproject.org/luajava/, 2006-05-12;

45. **LUA.org**,
www.lua.org, 2006-01-05;

46. **LUA.org**,
www.lua.org/uses.html, 2006-01-16;

47. **W3C Recommendation**, Extensible Markup Language (XML),
www.w3.org/TR/REC-xml/, 2007-08-08;

48. **wikipedia.org**,
de.wikipedia.org/wiki/Backus-Naur-Form, 2006-08-10;

49. **wikipedia.org**,
de.wikipedia.org/wiki/Logik, 2007-08-06;

50. **Michael A. Wirth**, Image Segmentation,
www.uoguelph.ca/~mwirth/CIS6320/lecture4.pdf, 2006-04-28.