Die approbierte Originalversion dieser Dissertation ist in der Hauptbibliothek der Technischen Universität Wien aufgestellt und zugänglich.
http://www.ub.tuwien.ac.at

TU UB WIEN Universitätsbibliothek

The approved original version of this thesis is available at the main library of the Vienna University of Technology.

http://www.ub.tuwien.ac.at/eng

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Numeric Methods for Configuration Management

DISSERTATION

zur Erlangung des akademischen Grades

Doktorin der technischen Wissenschaften

eingereicht von

Tanja Sisel
Matrikelnummer 0005351

an der Fakultät für Informatik
der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Gernot Salzer

Diese Dissertation haben begutachtet:

(Ao.Univ.Prof. Gernot Salzer) (Prof. Mira Balaban)

Wien, 27.08.2013 (Tanja Sisel)

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.ac.at

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Numeric Methods for Configuration Management

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktorin der technischen Wissenschaften

by

Tanja Sisel
Registration Number 0005351

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao.Univ.Prof. Gernot Salzer

The dissertation has been reviewed by:

(Ao.Univ.Prof. Gernot Salzer) (Prof. Mira Balaban)

Wien, 27.08.2013

(Tanja Sisel)

Erklärung zur Verfassung der Arbeit

Tanja Sisel
Schrekergasse 46/4/12, 1160 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum)	(Unterschrift Verfasserin)

Acknowledgements

First of all I want to thank the University of Technology of Vienna, who facilitated this thesis by funding the research as an innovative ideas ("Innovative Ideen") project.

Special thanks go to my "Doktorvater" (doctoral thesis supervisor) Gernot Salzer for placing confidence in me and for continuously supporting and encouraging me in my efforts. I also want to thank Ingo Feinerer for the numerous productive discussions (together with Gernot) that lead to the results we published together.

My gratitude goes to my parents Christina and Dieter, who have supported me (financially as well as emotionally) from the very beginning of my studies to the final version of this thesis. I love you!

Furthermore I want to thank my dear friends Gregor and Cordi for helping me to relax when I was stressed and tensed up. Thank you for being there for me! And of course thanks to everybody else who cheered me up and helped me get through my studies and this thesis.

Abstract

UML class diagrams play a central role in software engineering and are increasingly also used for modelling (product) configuration systems. Although important, formal reasoning about class diagrams is still rarely done. A main reason is that the expressive languages used to specify models have a high computational complexity, which makes formal reasoning time-consuming and expensive. Therefore we develop efficient methods tailored to specific tasks, focusing on a limited range of UML elements and constraints. With this approach we are able to guide the engineer through the design phase and to give immediate feedback after each change made to a diagram.

We start by considering the reduction of multiplicities. Multiplicities are intervals restricting the number of connected objects. Because of additional constraints (e.g. imposed by parallel association chains) it may happen that the bounds of the intervals cannot be reached. As this may be caused by some underlying misconception this situation should be detected and eliminated. We discuss bounds on the number of connected objects and develop a method for composing associations based on the translation of class diagrams to systems of linear inequalities. Furthermore, we introduce the concept of equality constraints that may lead to redundant multiplicities and develop a formula for reducing these multiplicities. Finally, we present an algorithm for reducing association chains and models with respect to equality constraints. We also discuss the effects of equality constraints on the satisfiability of a model and its minimal instances.

In configuration management we are not only interested in models, but also in generating instances (also called configurations). By solving the ILP problem formed by the system of inequalities we obtain the number of objects per class required for a minimal instance, as well as a range for the number of links for each association. In this thesis we present a method for distributing the links between the objects, such that the resulting configuration is an instance of the underlying model. We show how configuration completion and configuration repair can be addressed by solving minimum cost flow problems for certain flow networks derived from the model. This way we tackle the problem of reconfiguring legacy systems when requirements change.

Kurzfassung

UML Klassendiagramme spielen eine zentrale Rolle im Bereich Software Engineering und werden zunehmend auch für das Modellieren von (Produkt-)Konfigurationssystemen verwendet. Eine Überprüfung formaler Eigenschaften wie der Konsistenz findet trotz ihrer Relevanz in der Regel nicht statt, da diese Fragestellungen für die verwendeten Formalismen wegen deren Ausdrucksstärke von hoher Komplexität und daher zeitintensiv sind. In dieser Dissertation beschränken wir uns daher auf ausgewählte UML Sprachelemente, die eine effiziente Behandlung zulassen. Dadurch können Entwickler bereits während der Designphase durch kontinuierliche Rückmeldungen unterstützt werden.

In der vorliegenden Arbeit betrachten wir zunächst die Reduktion von Multiplizitäten. Multiplizitäten beschränken die Anzahl verbundener Objekte. Aufgrund anderer Einschränkungen (z.B. durch parallele Assoziationsketten) kommt es vor, dass diese Schranken gar nicht erreicht werden können. Dieser Umstand weist auf mögliche Modellierungsfehler hin und sollte daher vom System aufgezeigt werden. Wir entwickeln eine Methode um scharfe Multiplizitätsschranken zu berechnen. Basierend auf einer Übersetzung von Klassendiagrammen in ein lineares Ungleichungssystem beleuchten wir zuerst die Beschränkungen der Anzahl miteinander verbundener Objekte sowie die Bedeutung von zusammengesetzten Assoziationen. Danach präsentieren wir das Konzept von Assoziationsgleichungen (equality constraints), die zu redundanten Multiplizitäten führen können, und entwickeln eine Formel zur Reduktion dieser Multiplizitäten. Schlussendlich stellen wir einen Algorithmus zur Reduzierung von Assoziationsketten und Modellen vor. Wir diskutieren außerdem, welche Auswirkungen derartige Gleichungen auf die Erfüllbarkeit von Modellen und auf deren minimale Instanzen hat.

Im Configuration Management sind wir nicht nur an Modellen, sondern auch an der Generierung von Instanzen (auch Konfigurationen genannt) interessiert. Das Lösen des Ungleichungssystems liefert die Anzahl der minimal benötigen Objekte pro Klasse sowie ein Intervall für die Anzahl benötigter Links je Assoziation. Wir präsentieren eine Methode, mit der diese Links unter den Objekten so aufgeteilt werden können, dass die resultierende Konfiguration eine Instanz des zugrunde liegenden Modells ist. Durch das Lösen von Minimum Cost Flow Problemen in speziellen Flussnetzwerken lassen sich dabei verschiedene Kriterien berücksichtigen. Insbesondere sind Varianten dieser Netzwerke auch dazu geeignet, Konfigurationen zu vervollständigen und zu reparieren. Diese Probleme treten etwa bei der Rekonfiguration von Altsystemen auf, wenn sich Anforderungen ändern.

Contents

1 Introduction **1**
1.1 Motivation 1
1.2 Problem Statement and Aim of the work 2
1.3 Contributions 3
1.4 Structure of the work 3

2 Models and their instances **5**
2.1 Unified Modeling Language 5
2.2 Basic Definitions 7
2.3 Translation into linear inequalities 9

3 Configuration Management **13**
3.1 Definitions 13
3.2 Challenges in Configuration Management 14

4 Associations and Association Chains **21**
4.1 Connected objects 21
4.2 Composition of associations 26

5 Reducing Multiplicities and Models **31**
5.1 Motivation 31
5.2 Equations over association chains 32
5.3 Reducing multiplicities 34
5.4 Reducing Models 40

6 Effects of Equality Constraints **43**
6.1 Satisfiability under equality constraints 43
6.2 Minimal satisfying instance under equality constraints 46
6.3 Tree-generating equations 49

7 Linking Objects with Netflow Algorithms **53**
7.1 Motivation 53
7.2 Flow Networks and the Minimum Cost Flow Problem 54
7.3 Distributing Links 56

7.4 Completing Configurations 60
7.5 Repairing Configurations 63
7.6 Priority Links 65
7.7 Choosing the number of links 72
7.8 Different costs – different results 74
7.9 Can we get more for less? 75

8 Related Work **79**
8.1 Formalising UML Class Diagrams 79
8.2 Configuration Management and Reconfiguration 81
8.3 Detecting Redundancies 82

9 Conclusion **85**

Bibliography **89**

1 Introduction

> The beginning of knowledge is the discovery of something we do not understand.
>
> Frank Herbert

1.1 Motivation

This thesis was motivated by the limitations that our industrial partners experienced when using standard UML tools in the field of *configuration management*. Configuration management is the task of specifying admissible arrangements of functional units, finding setups that are optimal according to some criteria, and maintaining them when requirements change. Functional units may be software or hardware components or physical objects like computers or railway stations. The specification of these arrangements can for example be done with UML class diagrams, which basically consist of classes (which abstract components) and associations between them (specifying the relation between the components). Class diagrams are mainly used in software engineering, hence existing tools are especially designed to meet the needs of software engineers. As configuration management has requirements different from software engineering, the functionality offered by current UML tools does not suffice.

In contrast to software engineering, where instances are only generated at runtime and in general do not exist independently from the model, instances in configuration management have a life of their own. A house that has been built according to a specification may still exist long after the specification has been erased. In software engineering the question of finding the minimal instantiation of a specification that satisfies all constraints is rarely asked, whereas in configuration management finding a setup with fewer components is usually preferable. Furthermore, instances shall be adapted with as few rearrangements as possible, if requirements change. If, for example, components shall be added to a computer network, we will try to keep the existing cables instead of rewiring the whole network to save costs. Another difference between software engineering and configuration management is the variety of the so-called multiplicity constraints imposed on the associations occurring in the diagrams. Multiplicities are intervals of positive integers $n..N$ specifying that each instance of a particular class has to be linked to at least n and at most N instances of some other class. In software engineering they are mainly of the form $0..\star$ (unrestricted), $1..\star$ (at least one), $0..1$ (at most one, optional), and $1..1$ (exactly one). Specifications in configuration management, on the other hand, may as well state that e.g. a computer

has to be connected to at least one and at most four printers, thus leading to a higher variability regarding the multiplicities.

While UML tools offer the possibility to create and maintain diagrams (i.e. specifications) and to automatically generate code, they do not offer any means to handle instances (i.e. configurations). The possibility to create and maintain instances, to check them against specifications and to check the specifications themselves for inconsistencies is completely missing.

These unaddressed problems lead to a research programme that was started by Gernot Salzer and Ingo Feinerer. Based on their basic results on formal verification of UML class diagrams (see [17,20,21]), we developed theoretical foundations and core algorithms that can be integrated into a tool offering the functionality required for configuration management.

1.2 Problem Statement and Aim of the work

The higher variability of multiplicities in configuration management permits over-restrictive constraints, leading to unsatisfiable specifications, i.e. specifications that are not satisfiable by any non-trivial configuration (= instance). In complex diagrams the reasons for the unsatisfiability of a specification are not easy to detect. We would therefore like to be able to check the satisfiability of specifications automatically.

Furthermore, we would like to create optimal configurations (e.g. containing the least number of objects) directly from the specification and to check whether a given configuration (for example created by the user) satisfies all constraints imposed by the corresponding specification. A routine performing this *instance checking* should as well provide feedback on the source of the error, if a configuration is no satisfying instance of the specification.

To be able to maintain an existing system when requirements change, we need to find a conservative extension of a configuration. This means that we want to keep as many existing objects and links as possible while constructing a configuration that satisfies all constraints imposed by the (changed) specification. We call this problem the *minimal repair* problem.

For several real-life problems, we additionally need constraints to express that two parallel relations (or chains of relations) lead to the same objects for each object of the starting class. This is, for example, used to specify that the belongings of a specific person may only be stored in his own storage. These additional constraints may influence the number of possible partner objects for several classes that are part of this constraint. Hence, we might not be able to reach the bounds of some multiplicities. As this hints at some underlying misconception, these multiplicities should be detected and tightened (*reducing multiplicities*). Moreover, additional constraints may as well influence the satisfiability of a specification. Therefore, we need to check whether a specification is still satisfiable after adding such constraints (*checking E-satisfiability*).

Prior to this thesis, Gernot Salzer and Ingo Feinerer developed a numerical approach solving some of the described problems. Their approach is to translate UML class diagrams into linear inequalities and to use standard *Integer Linear Programming* (ILP) solvers. With this technique we can check the satisfiability of a specification, compute the number of objects and links necessary to build a minimal instance of the specification, and check whether a given configuration satisfies all multiplicity constraints.

This thesis extends this approach. We investigate the influence of equality constraints and develop methods for minimising multiplicities, for finding link distributions, and for completing or repairing configurations. We use numerical reasoning to solve the problems. We compute (minimal) instances and check the satisfiability of diagrams with ILP. For (re-)linking objects (i.e. completing and repairing configurations) we use netflow algorithms.

1.3 Contributions

This thesis is part of a joint project with Gernot Salzer and Ingo Feinerer[1]. Many results were developed or at least refined by all of us together in long discussions.

The basic method of reducing multiplicities was developed by Ingo Feinerer and Gernot Salzer. Tanja Sisel improved and generalised the result and implemented the procedure in a prototype called CLEWS[2] [34]. This method was published at MODELS 2011 [22], with Tanja Sisel presenting the results at the conference in Wellington, New Zealand.

The method of linking objects and repairing configurations via flow algorithms (see Chapter 7) was mainly developed by Tanja Sisel and Ingo Feinerer (Flow networks were initially used to distribute links uniformly between the objects of two classes [34]). Gernot Salzer contributed to the formal presentation of the results and gave valuable feedback during the development of the method. The results were published at ISMIS 2012 [19]. Tanja Sisel presented the work at the conference in Macao, China. The more detailed concept of priority links presented in this thesis was developed by Tanja Sisel. The considerations on how to choose the number of links and the costs on the arcs, and on the more-for-less paradox were also done by Tanja Sisel.

The discussion of equations over association chains and their effects, as well as the concept of tree-generating equations (see Chapter 6) were published at TASE 2013 [23]. Tanja Sisel, Gernot Salzer and Ingo Feinerer contributed proportionally to the development of the theoretical background, the used examples and the composition of the paper.

A further contribution of this thesis is the systematic and uniform presentation of all results obtained within the project throughout the last four years, including proofs and examples that had to be omitted from the papers for space reasons.

1.4 Structure of the work

This thesis is structured as follows.

Chapter 2 gives basic definitions and semantics of models (i.e. UML class diagrams) and their instances.

In Chapter 3 we explain what *Configuration Managements* is about and which problems occur in this domain, as well as our research programme.

Chapter 4 investigates properties of associations and their instantiations, *relations*, and introduces the concept of associations chains.

[1] Gernot Salzer is the supervisor of this thesis, Ingo Feinerer was also involved in the development process.

[2] CLEWS: Class Editor With Semantics (www.logic.at/clews), originally developed by Gerhard Niederbrucker, a former master student advised by Gernot Salzer.

Based on these properties, we develop a method for reducing multiplicities in Chapter 5. For this purpose we first introduce *equality constraints*, and afterwards give a formula and a procedure for tightening multiplicities within an association chain. Finally, we show how to reduce models.

In Chapter 6 we consider the effects of adding equality constraints to a model. With the help of examples we show why equality constraints influence the satisfiability of a model and how to check the *E-satisfiability* of a model. We introduce a particular family of models, so-called *tree-generating models*, and explain important properties of these models.

In Chapter 7 we explain how to use netflow algorithms to distribute links between objects and how to extend and repair configurations. We also discuss how to set the costs of the arcs accordingly and introduce different kinds of *priority links*.

Chapter 8 discusses related work and Chapter 9 critically reflects on the results of this thesis, summarises them and gives an overview of open questions.

2 Models and their instances

> The greatest challenge to any thinker is stating the problem in a way that will allow a solution.
>
> Bertrand Russel

We start with a short introduction to UML class diagrams, which we use to model our problems. After defining some basic concepts, we show how models are translated into systems of inequalities and how we can check satisfiability and construct minimal instances with the help of these inequalities.

2.1 Unified Modeling Language

The *Unified Modeling Language* [36], UML, is a standardised modelling language in the form of a collection of different diagram types. Diagrams are expressive and ideal to exchange ideas and (domain) knowledge. Therefore UML is widespread and well-known in the domain of software engineering and has a broad tool support. The goal of UML is to cover a broad range of application areas. The provided diagram types meet various modelling demands. An overview can be found in [30]. In this thesis we focus on UML class diagrams.

Class diagrams serve to abstract objects and their relationships. Typical application areas of class diagrams are the design of databases and the design of object oriented software.

A class diagram basically consists of *classes* related by *associations*. UML class diagrams provide several more features, but we will focus on these basic components, as these suffice for many applications. A complete feature overview can be found in [36].

Classes model types of objects (or entities like "address") that have specific attributes in common. A class is identified by its name, for example "Person". Additionally it has one or more attributes, like "Name", "Age", and so on. Every instance of a class (for example in the form of a record in a database or in the form of a physical object) has a concrete value for each attribute, e.g. "Tom". These attribute values characterise the instance.

Associations model relationships between objects represented by classes. An association can be binary or n-ary, depending on the number of classes it relates. The most common one is the *binary association*. A binary association $C\ {}_{m..M}{\Rightarrow}_{n..N}\ D$ as depicted in Figure 2.1 consists of two classes, C and D, an association name u, and multiplicities and uniqueness constraints at each association end. The arrow indicates the orientation of the association. The multiplicities are intervals over integers. The interval $n..N$ restricts the number of partner objects (or the

number of links to partner objects) for each object of class C and the interval $m..M$ those for each object of class D, as UML adheres to a look-across policy. The so-called uniqueness-attributes γ_1 and γ_2 can either be $unique$ (which is the default in UML) or $non\text{-}unique$. The multiplicities of an association end that is labelled $unique$ restrict the number of partner objects, whereas those of an association end labelled as $non\text{-}unique$ restrict the number of links to partner objects.

Additional constraints, like lower bounds on classes, can be defined using the *Object Constraint Language* (OCL, see [37]). Nevertheless, we will introduce our own notation to specify the few additional constraints we need. On the one hand OCL constraints are rather bulky, and on the other hand we only need a tiny fragment of OCL not justifying the introduction of another formalism.

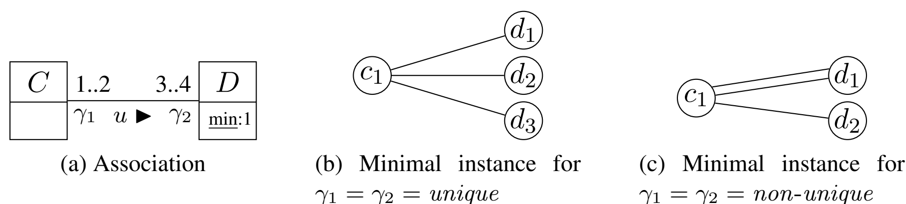
Lower bounds on classes give the minimally required number of objects for a class. A lower bound of a_{min} on class C means that in every valid instance of this model there have to be at least a_{min} objects of class C. Lower bounds are non-negative integer values. We will specify the lower bound on the number of objects of a class by a static class attribute `min` (see Figure 2.2a), instead of expressing it by an OCL constraint.

Figure 2.1: Naming conventions for binary associations.

Example 2.1. Figure 2.2 illustrates the effect of the multiplicity attributes $unique$ and $non\text{-}unique$. The model of the association in Figure 2.2a requires that there is at least one D-object, as stated by the static class attribute `min`.

In Figure 2.2b the multiplicities on both association ends are tagged as $unique$. Starting with d_1, we need at least one C-object, c_1 because of multiplicity $1..2$, which in turn needs at least three D-objects, d_1, d_2, and d_3.

In Figure 2.2c, the multiplicities are $non\text{-}unique$. Starting again with the required object d_1, we need at least one link to a C-object, c_1, which in turn needs at least three links to D-objects. Since d_1 can take another link, it suffices to add a second D-object.



(a) Association

(b) Minimal instance for $\gamma_1 = \gamma_2 = unique$

(c) Minimal instance for $\gamma_1 = \gamma_2 = non\text{-}unique$

Figure 2.2: Binary association with minimal solutions for unique and non-unique ends

2.2 Basic Definitions

We now introduce the basic concepts we need throughout the rest of this thesis.

Remark 2.2. The following definitions and chapters only consider $unique\text{-}unique$ associations, which are the default in UML. We therefore ignore the uniqueness-attributes described in the previous section and assume that all associations are tagged as $unique\text{-}unique$. Associations tagged as $nonunique\text{-}nonunique$ will only be considered in Chapter 7.

Definition 2.3 (Model)**.** A *model* $\mathfrak{M} = (\mathcal{C}, \mathcal{A})$ consists of a set $\mathcal{C}$ of classes and a set $\mathcal{A}$ of binary associations. Classes and associations are represented by unique names. Each association $u \in \mathcal{A}$ is of a specific *type*, $\text{type}(u)$, of the form $C\ {}_{m..M}{\Rightarrow}_{n..N}\ D$, where C and D specify the classes related by the association, and m, M, n, and N are natural numbers with $0 \leq m \leq M$, $0 \leq n \leq N$ and $M, N \geq 1$. Expressions of the form $n..N$ are called multiplicities and will be interpreted as intervals. The values n and m are called *lower bounds*, N and M are called *upper bounds*. The classes related by an association u are called the *signature* of u: $sig(u) = (C, D)$.

Upper bounds may also be the symbol $*$ denoting infinity or more precisely an arbitrary, but finite value. The multiplicity $0..*$ may be omitted. Figure 2.1 shows the graphical representation of an association u that is of type $C\ {}_{m..M}{\Rightarrow}_{n..N}\ D$.

Later on we will extend the definition of a model by equations.

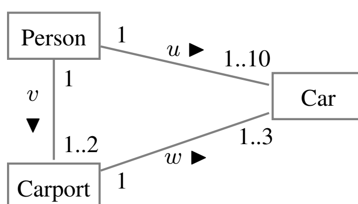


Figure 2.3: Model of persons, their cars and carports.

Example 2.4 (Carport Example)**.** Consider the model $\mathfrak{M}$ in Figure 2.3 where each person can have between one and ten cars and one or two carports, each carport has place for one to three cars and each car has to be parked in a carport. For $\mathfrak{M}$ we have $\mathcal{C} = \{\text{Car}, \text{Carport}, \text{Person}\}$, $\mathcal{A} = \{u, v, w\}$ with u: Person ${}_{1..1}{\Rightarrow}_{1..10}$ Car, v: Person ${}_{1..1}{\Rightarrow}_{1..2}$ Carport, and w: Carport ${}_{1..1}{\Rightarrow}_{1..3}$ Car.

A class defines the main characteristics of a specific type of objects (e.g. cars), whereas an association specifies the properties of the possible connections (i.e. links) between the objects. A *link* connects two objects belonging to distinct classes. Objects and the links connecting them together form an instance.

Definition 2.5 (Instance)**.** An *instance* $\mathfrak{I} = (\mathcal{O}, \mathcal{L})$ consists of a set $\mathcal{O}$ of objects and a set $\mathcal{L}$ of links, with objects being represented by unique names. A link $(o, p) \in \mathcal{L}$ is an ordered pair of objects related by the link.

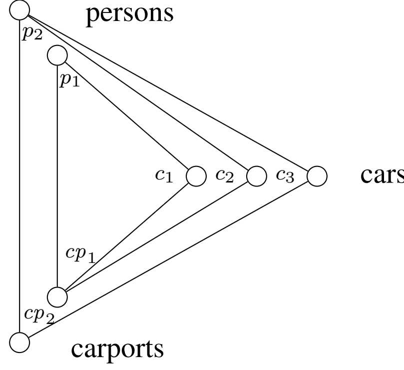


Figure 2.4: Instance of the model in figure 2.3.

Example 2.6. The instance $\mathfrak{I} = (\mathcal{O}, \mathcal{L})$ in Figure 2.4 consists of $\mathcal{O} = \{p_1, p_2, c_1, \ldots, c_8, cp_1, cp_2, cp_3\}$ and $\mathcal{L} = \{(p_1, c_1), (p_2, c_2), (p_2, c_3), (p_1, cp_1)(p_2, cp_2)(cp_1, c_1), (cp_1, c_2), (cp_2, c_3)\}$.

For some definitions, problem descriptions and problem solutions, it is convenient to partition the set of links into *relations*.

Definition 2.7 (Relation)**.** A relation $\mathrm{rel}(u)$ or r_u is the set of all links (i.e. object pairs) instantiating an association u. Hence, each relation of an instance $\mathfrak{I} = (\mathcal{O}, \mathcal{L})$ is a subset of $\mathcal{L}$. Formally, $\mathrm{rel}(u) = assoc^{-1}(u)$. We say that $\mathrm{rel}(u)$ is of type $C \;{}_{m..M}\!\Rightarrow_{n..N}\; D$ if u is of this type.

In general we assume that every instance is an instance of some model.

Definition 2.8 (Instance of a model)**.** $\mathfrak{I}$ is an *instance of a model* $\mathfrak{M}$, if it is well-typed regarding $\mathfrak{M}$. This means:

- Every object $o \in \mathcal{O}$ is an instance of exactly one class $C \in \mathcal{C}$, written as $\mathrm{class}(o) = C$; we say that o is a C-object.
- Every link $l \in \mathcal{L}$ is an instance of exactly one association $u \in \mathcal{A}$, written as $assoc(l) = u$; we call l a u-link.
- The links are well-typed: $\mathrm{class}(l) = sig(assoc(l))$ holds for all links $l \in \mathcal{L}$. (The function class is extended to links by $\mathrm{class}(l) = \mathrm{class}((o, p)) = (\mathrm{class}(o), \mathrm{class}(p))$.)

We also use the class name C to refer to the set $\mathrm{class}^{-1}(C)$ of all C-objects; hence $o \in C$ is another way of saying that o is a C-object. Accordingly, $|C| = |\mathrm{class}^{-1}(C)|$ denotes the number of C-objects.

Example 2.9. The instance $\mathfrak{I}$ in Example 2.6 is an instance of the model $\mathfrak{M}$ in Example 2.4 if we set $\mathrm{class}(p_i) = \text{Person}$, $\mathrm{class}(c_i) = \text{Car}$, $\mathrm{class}(cp_i) = \text{Carport}$, and $assoc(p_i, c_j) = u$ for $i = 1 \ldots 2$ and $j = 1 \ldots 3$, $assoc(p_i, cp_j) = v$ for $i, j = 1 \ldots 2$, and $assoc(cp_j, c_i) = w$ for $j = 1 \ldots 3, i = 1 \ldots 2$. Note that the links are well-typed; we have e.g. $\mathrm{class}((p_1, c_1)) = (\text{Person}, \text{Car}) = sig(u) = sig(assoc(p_1, c_1)))$.

In the following chapters a central concept is the one of instances satisfying a specific model. This is the case, if an instance respects all constraints imposed by the corresponding model. For specifying whether multiplicity constraints are respected, we need to define the number of connected objects: Let $\gamma_D(o) = |\{\, p \in D \mid (o,p) \text{ for some } (o,p) \in \mathcal{L}\,\}|$ be the number of D-objects linked to the C-object o, and similarly let $\gamma_C(p)$ be the number of C-objects linked to the D-object p.

Definition 2.10 (Satisfying instance)**.** An instance $\mathfrak{I}$ *satisfies* a model $\mathfrak{M}$ if it is an instance of $\mathfrak{M}$ and if all links respect the multiplicities of their association: For every association $u \in \mathcal{A}$ of type $C\ {}_{m..M}{\Rightarrow}_{n..N}\ D$ and all objects $o \in C$, $p \in D$ we have $n \le \gamma_D(o) \le N$ and $m \le \gamma_C(p) \le M$.

Note that this definition corresponds to the multiplicity attribute $unique$, which is the default in UML [36]. For a discussion of further attributes see [21, 22].

Example 2.11. The instance $\mathfrak{I}$ in Example 2.6 is a satisfying instance of the model $\mathfrak{M}$ in Example 2.4 as it is an instance of $\mathfrak{M}$ (see Example 2.9) and since all links respect the multiplicities. E.g., for association w and object cp_1 we have $n = 1 \le \gamma_{\text{Car}}(cp_1) = |\{c_1, c_2\}| = 2 \le 3 = N$.

An interval $a..A$ is *tighter* than another interval $b..B$, if either $a > b$ and $A \le B$ or $a \ge b$ and $A < B$ holds. A type T is weaker than or equal to a type T' (T' is stronger than or equal to T), if every relation of type T' is also of type T. Since we do not consider hierarchies on classes in this context, type T is weaker if its intervals contain those of T' (i.e. the intervals of type T' are tighter than those of type T). This leads to the following lemma, which is used implicitly throughout this thesis.

Lemma 2.12. *Let r_u be of type $C\ {}_{m..M}{\Rightarrow}_{n..N}\ D$. Then r_u is also of type $C\ {}_{m'..M'}{\Rightarrow}_{n'..N'}\ D$ for all $m' \le m$, $n' \le n$, $M' \ge M$, and $N' \ge N$.*

Example 2.13. The association v in Figure 2.5b has tighter bounds than association u in Figure 2.5a. Hence, every instance of v is also an instance of u, but not vice-versa. The instance in Figure 2.6a satisfies both associations, whereas the one in Figure 2.6b only satisfies association u. The second instance violates the multiplicity 2..2 from association v, as object d_1 is only connected to a single C-object.

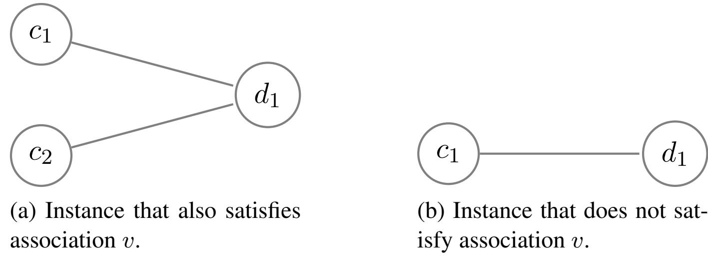
(a) Association u.

(b) Association v is of a stronger type than u.

Figure 2.5: Models of two different associations between classes C and D.

2.3 Translation into linear inequalities

To allow a formal examination of models, we need to formalise them. One possible formalisation of class diagrams is a translation to a system of inequalities (see [20, 32]). Each association is



(a) Instance that also satisfies association v.

(b) Instance that does not satisfy association v.

Figure 2.6: Instances that satisfy association u from Figure 2.5.

translated independently from all other associations. A binary association as shown in Figure 2.1 is translated to the inequalities

$$Nx \geq my \qquad My \geq nx \tag{2.1}$$
$$x \geq c \qquad y \geq d \tag{2.2}$$
$$xy \geq my \qquad xy \geq nx \tag{2.3}$$

where x and y are variables for the cardinalities of the classes C and D. The multiplicity constraints are represented by the inequalities (2.1), while the inequalities (2.2) model the lower bounds on the classes. The inequalities (2.3) ensure that there are enough objects to satisfy the $unique$ constraint. A model has a satisfying instance if and only if the inequalities are solvable [21, 32].

The number ℓ of links required to build a satisfying instance is bounded by

$$\max(nx, my) \leq \ell \leq \min(Nx, My, xy) \ .$$

Computing instances using ILP

The inequalities we obtain from the translation form a so-called ILP program. By solving it we can efficiently check the satisfiability of the model (i.e. we can check whether a satisfying instance exists). At the same time, we obtain the number of objects and links required for each class and association. Furthermore we know that an instance satisfies a model if the class cardinalities satisfy the inequalities corresponding to the model. The translation is correct and complete: For every satisfying instance there exists a corresponding solution of the inequalities, and for every solution of the latter we can construct a valid satisfying instance by adding appropriate links (see [21] for a detailed discussion). Since the ILP solutions are closed under linear combinations and the minimum operator, the minimal solution is unique.

For further applications of this approach (for instance generalisation to multiary associations and other uniqueness attributes) see [16, 20].

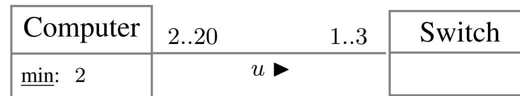


Figure 2.7: Example of a model: Every computer has to be connected to 1 to 3 switches, and each switch must be connected to at least two and at most 20 computers. Each instance satisfying this model has to consist of at least two computers.

Example 2.14. The model in Figure 2.7 corresponds to the inequalities

$$\begin{aligned} 3x &\geq 2y & \qquad 20y &\geq 1x \\ x &\geq 2 & y &\geq 0 \\ xy &\geq 2y & xy &\geq 1x \end{aligned}$$

and the number of links is bounded by $\max(1x, 2y) \leq \ell \leq \min(3x, 20y, xy)$. The minimal solution of the inequalities is $x = 2$, $y = 1$ and we need at least $\ell = 2$ links. This means that we need at least two computers, one switch and two links between them.

So the ILP program above tells us whether a model admits any instances, and if so, how many objects and links we need for a minimal instance. We now need to find a concrete distribution of the links to build a complete satisfying instance of the model. Distributing the required number of links uniformly among the objects yields one possible instance (see the discussion on *balanced sequences* in [21]). A more flexible approach of distributing links is described in Chapter 7.

3 Configuration Management

> If the only tool you have is a hammer, you tend to see every problem as a nail.
>
> Abraham Maslow

3.1 Definitions

Configuration management is the task of modelling, building and maintaining real life systems. The components of such systems are functional units, which may be software, hardware or other physical objects like computers or train stations. In configuration management we want to specify admissible arrangements of functional units, set them up according to some criteria of optimality and maintain them when requirements change.

A *specification* fixes general properties of objects and their relation to each other. In software engineering, a specification is called a *model*. In our context specifications consist of a set of classes $\mathcal{C}$, a set of binary associations $\mathcal{A}$, and later on also of a set of equations $\mathcal{E}$.

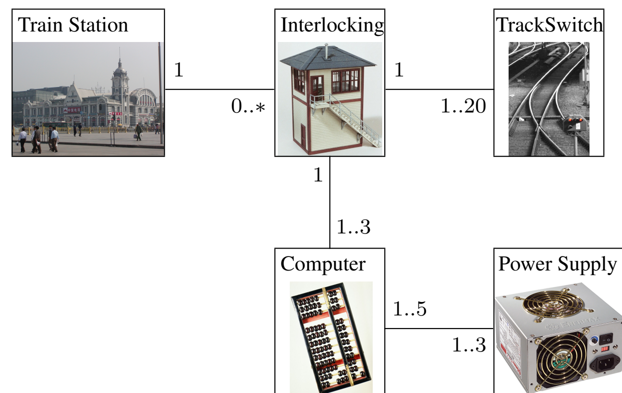
A *configuration* in our sense is a collection of objects, partly related to each other, i.e. connected by links. Configurations represent physical multi-part systems, like railway-systems, computers or cars. Configurations are represented by a set of objects $\mathcal{O}$ and a set of links $\mathcal{L}$. A link is an ordered pair of objects, e.g. (c_1, d_2). A configuration can be an instance of some specification. In software engineering, configurations correspond to instances.

A configuration is an *instance of a specification*, if each object is associated with a specific class and each link is associated with a specific association, which corresponds to the definition of "Instance of a model" (see Section 2.2).

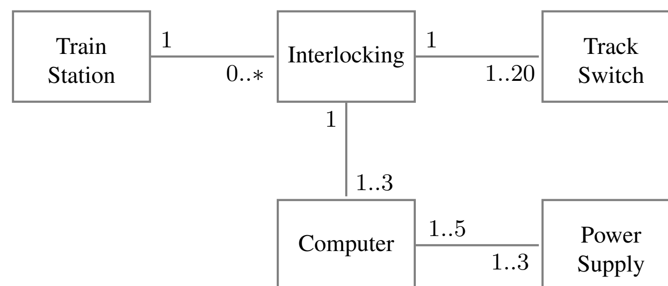
In configuration management configurations, unlike most instances in software engineering, exist independently from any specification, as they consist of real-life, physical objects or functional units. As an example, consider a railway system built to satisfy a particular specification. Even if the specification changes or does not exist anymore, the railway system with all its components like train stations and rails remains.

Hence, it is necessary to maintain configurations, even if requirements change.

Example 3.1. Consider the railway control system depicted in Figure 3.1a. It consists of physical objects like train stations and computers. We model this system as a UML class diagram. From this specification (see Figure 3.1b) we want to generate concrete instances that can then



(a) The railway control system components modelled as classes and their relations modelled as associations.



(b) UML Class Diagram of the railway control system.

Figure 3.1: Railway Control System

be realised in real life. Furthermore, we want to check whether existing railway control systems adhere to the corresponding specification and find ways to correct systems that do not comply with the specification.

3.2 Challenges in Configuration Management

We now want to introduce our research programme by describing important challenges in the domain of configuration management that were identified in [16] and [21].

The goal of our research is to find solutions to those problems connected with configuration management and to provide algorithms that can be combined in a tool.

Usability

The overall intention is to maintain *usability* by hiding formal methods behind familiar user interfaces, making formal specifications as intuitive as possible and providing comprehensible feedback in cases of error. Formal methods, as useful as they are, have the problem of being difficult to use. Most modellers are not familiar with them and furthermore do not have the time to learn and apply them while modelling. Hence, it is preferable to hide the formal methods

behind a familiar user interface and to apply them in the background. A first step to solve this challenge is to model specifications with UML class diagrams. Class diagrams are well-known and hence do not force the users to deal with a new modelling formalism. We translate the diagrams into inequalities in the background and do the formal reasoning hidden from the user. Finally, we give feedback by highlighting errors or suggesting optimisations directly in the diagram. Additionally, we provide a graphical interface to model configurations (instances). This feature gives us the possibility to give feedback on errors in instances as well.

All problems described below should be solved having usability in mind.

Instance Checking

INSTANCE
Input: A model $\mathfrak{M}$, an instance $\mathfrak{I}$, and mappings class and assoc.
Question: Is the configuration $\mathfrak{I}$ a satisfying instance of the model $\mathfrak{M}$ with respect to class and assoc?

This problem is basically solved by checking whether a given configuration satisfies all constraints specified by the corresponding model $\mathfrak{M}$. The main constraints are multiplicities and lower bounds of classes, but there can also be further constraints like equations over chains of associations (see Chapter 5).

Usually checking constraints is computationally easy, as it can be done locally (i.e. whether constraints like multiplicities are satisfied can be checked for each association separately). Instance-checking can therefore be done automatically after each change by the user, as it only has to be done for the affected parts. Thus, we can give immediate feedback to the user directly in the instance by highlighting any parts that violate the model.

Example 3.2. Figure 3.3a is a satisfying instance of the model in Figure 3.2, if we set

- $\mathrm{class}(p_1) =$ Person
- $\mathrm{class}(c_i) =$ Computer for $i = 1 \dots 3$
- $\mathrm{class}(pr_i) =$ Printer for $i = 1 \dots 2$
- $\mathrm{assoc}(p_1, c_i) = u$ for $i = 1 \dots 3$
- $\mathrm{assoc}(c_i, pr_j) = v$ for $i = 1 \dots 3, j = 1 \dots 2$.

If we add a fourth computer to the instance (see Figure 3.3b) it is no satisfying instance of the model anymore, because person p_1 now has four computers, which are too many.

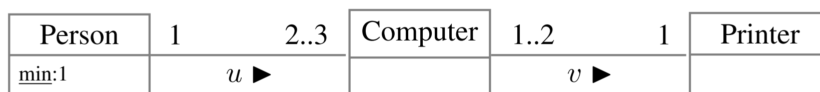
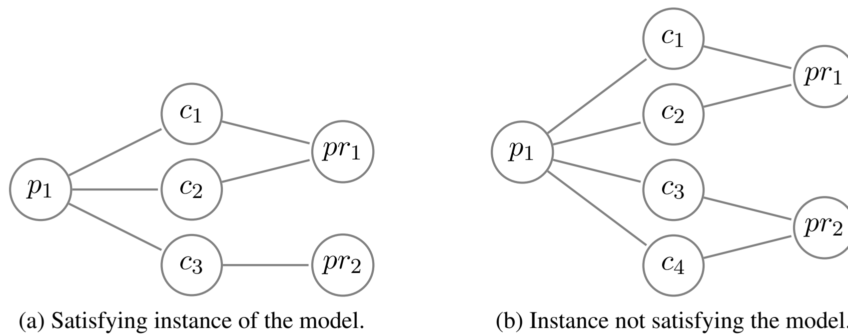


Figure 3.2: Model of persons, their computers and printers.



(a) Satisfying instance of the model.

(b) Instance not satisfying the model.

Figure 3.3: 2 instances of the model in Figure 3.2.

Satisfiability and Consistency

SATISFIABILITY
Input: A model $\mathfrak{M}$.
Question: Is there an instance satisfying $\mathfrak{M}$?

A model is satisfiable, if the constraints specified by the model admit at least one satisfying instance. For some constraints this problem might be of a high complexity. In description logics, for example, checking the satisfiability of constraints is exponential or sometimes even undecidable. As our goal is to develop a usable system, we decided to use a numerical approach, which can only deal with constraints of limited expressivity, but allows us to give immediate feedback. Checking satisfiability can be done in polynomial time by building a graph from the inequalities and running a variant of the all-pairs-shortest-path algorithm by Floyd-Warshall. A detailed description can be found in [21].

Example 3.3. The model in Figure 3.4 is unsatisfiable. To satisfy association u we need twice as many D-objects as C-objects and at the same time we need twice as many C-objects as D-objects to satisfy association v. Hence we cannot find any non-trivial instance (i.e. an instance consisting of at least one object of some class) satisfying both associations simultaneously.

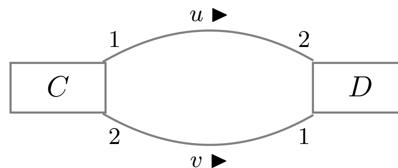


Figure 3.4: Unsatisfiable model of two associations between the same classes.

CONSISTENCY
Input: A model $\mathfrak{M}$.
Question: For each class C, is there an instance satisfying the model $\mathfrak{M}$ that contains at least one object of class C?

An inconsistent class diagram is a diagram, where for some particular class there exists no satisfying instance with at least one object of this class. Usually this is not intended and therefore hints at some underlying error. Such situations occur because of over-restrictive constraints. By adding a lower bound of (at least) one to each class in turn, checking consistency can be reduced to checking satisfiability.

Example 3.4. The model in Figure 3.5 is inconsistent. The two associations between classes C and D are unsatisfiable (see Example 3.3), but we still can find an instance satisfying the model. As we have a lower bound of 0 on the number of D-objects connected to each E-object via association w, an instance consisting only of E-objects is a non-trivial satisfying instance of the model. The model is inconsistent, because we cannot find satisfying instances containing any objects of classes C or D.

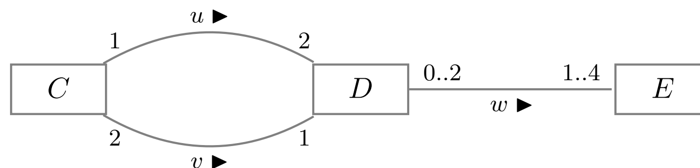


Figure 3.5: Inconsistent model of three classes.

Minimal Instance Problem

MINIMAL INSTANCE
Input: A model $\mathfrak{M}$ and an ordering $<$ on instances.
Output: An instance satisfying $\mathfrak{M}$ that is minimal w.r.t. ordering $<$.

To solve this problem, we need to find an instance of a given model that is minimal in some sense. As configuration management deals with real-life systems, we want to reduce costs, hence we search for the instance with lowest cost. Preferable instances might be the ones with the least objects in total or per class. Furthermore, objects of certain classes might be more cost effective than others and we therefore might prefer to choose more of them. We might as well want to minimise the number of links between the objects or find the most cost-effective link-distribution.

From the ILP program resulting from the translation of the model to inequalities we can compute the minimally needed number of objects and links for each class and association. We can even assign costs to classes and still get an optimal solution from the ILP solver. This is one of the main advantages of our numerical approach.

Example 3.5. The instance of the model in Figure 3.2 that is minimal regarding the number of objects is shown in Figure 3.6. It consists of one person p_1, two computers c_1 and c_2 and one printer pr_1.

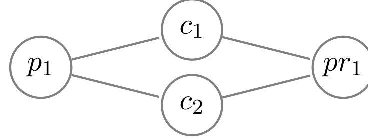


Figure 3.6: Minimal instance of the model in Figure 3.2.

Minimal Repair

MINIMAL REPAIR
Input: A model (specification) $\mathfrak{M}$, an instance (configuration) $\mathfrak{I}$ (in general not satisfying $\mathfrak{M}$), an ordering $<$ on instances, and a notion of similarity of instances.
Output: A $<$-minimal instance (configuration) $\mathfrak{I}'$ satisfying $\mathfrak{M}$ that is similar to $\mathfrak{I}$.

Modifying physical systems (i.e. configurations) costs money. We therefore want to find a conservative extension of the system, if requirements (i.e. the underlying specification) change. In this case it is preferable to keep as many components and links from the old configuration as possible to keep the adaption costs minimal. If, for example, a computer network needs to be adapted because the minimally required number of computers changes, it is preferable to leave the existing parts of the network as untouched as possible. A complete re-wiring of the cables of the network would not be the best solution, even if the conservative extension leads to a configuration that is not minimal in an absolute sense.

There is no unique definition of similarity of instances. It might, for example, be preferable to change (add/delete) as few objects and links as possible or it might even be forbidden to delete specific links (or any links at all).

This problem is discussed in more detail in sections 7.4 and 7.5.

Minimal Multiplicites

MINIMAL MULTIPLICITIES
Input: A model $\mathfrak{M}$.
Output: A model $\mathfrak{M}'$ equivalent to $\mathfrak{M}$, whose multiplicities are minimal among all models equivalent to $\mathfrak{M}$.

It can happen that the bounds of some multiplicity $n..N$ of a model are not reachable due to other restrictions. In such cases, we can tighten the interval of these multiplicities to $n'..N'$, with $n' \geq n$ and $N' \leq N$. This reduction does not affect the instances satisfying all constraints imposed by the model (i.e. the reduced model is equivalent to the original model).

Non-minimal multiplicities mirror the modeller's fuzzy perception of the system he wants to model and might even hint at some underlying misconception. Furthermore, reducing multiplicities can also help to reveal that a model is not satisfiable when taking some equality constraints into account. Hence, among all equivalent models we want to find the one with minimal multiplicities for all associations.

We deal with this problem in Chapter 5.

4 Associations and Association Chains

The important thing in science is not so much to obtain new facts as to discover new ways of thinking about them.

Sir William Bragg

To get a feeling for associations and relations, we now investigate some basic properties of associations and their corresponding relations and give some characteristics of the solutions of the inequalities from the previous chapter. Furthermore we introduce the concept of composing association chains. We partly published the results of this chapter in [23].

4.1 Connected objects

This section investigates the bounds on the number of objects connected to a specified number of objects of another class via a specific *unique-unique* association.

For the following observations, consider the following binary association with the corresponding inequalities (see (2.1) and (2.3) - we omit the lower bounds on classes for this purpose).

$$Nx \geq my \qquad My \geq nx \tag{4.1}$$

$$xy \geq my \qquad xy \geq nx \tag{4.2}$$

Note that the expression $xy \geq nx$ is equivalent to $y \geq n \cdot \operatorname{sgn}(x)$[1] or $y \geq n$ for $x > 0$. We assume that $m \geq 1$ and $n \geq 1$, as we are only interested in D-objects that are connected to at least one of the C-objects and vice versa. Hence, we set $m = 1$, if $m = 0$, which corresponds to using $\max(m, 1)$ instead of m in the inequalities. The same holds for n.

First of all, we want to determine the number of D-objects, y, connected to a given number x of C-objects via association u, where x is the total number of C-objects.

From the inequalities (4.1) we get

[1]The signum (or sign) function is defined by $\operatorname{sgn}(x) = 1$ for $x > 0$, $\operatorname{sgn}(0) = 0$, and $\operatorname{sgn}(x) = -1$ for $x < 0$.

$$\lceil \frac{my}{N} \rceil \leq x \leq \lfloor \frac{My}{n} \rfloor \quad (4.3)$$

$$\lceil \frac{nx}{M} \rceil \leq y \leq \lfloor \frac{Nx}{m} \rfloor \ . \quad (4.4)$$

As the number y of D-objects connected to a given number x of C-objects has to satisfy this inequality as well as $y \geq n \cdot \mathrm{sgn}(x)$, it is bounded by

$$\max(\left\lceil \frac{nx}{M} \right\rceil, n \cdot \mathrm{sgn}(x)) \leq y \leq \left\lfloor \frac{Nx}{m} \right\rfloor \quad (4.5)$$

The inverse direction (regarding the number of partner objects of class C for a given number of D-objects) is symmetric. We will use $\Delta_{n,M}(x)$ to abbreviate the lower bound $\max(\lceil \frac{n \cdot x}{M} \rceil, n \cdot \mathrm{sgn}(x))$.

Example 4.1. Consider the model in Figure 4.1. The instance in Figure 4.2a shows the maximal number of connected D-objects, y, for two C-objects (c_1 and c_2), which is $y = 4$. This corresponds to the upper bound for y in (4.5) for $x = 2$. We cannot connect more objects of class D to the C-objects, because each D-object needs at least two connected C-objects.

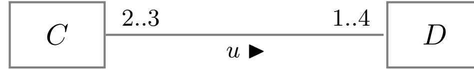


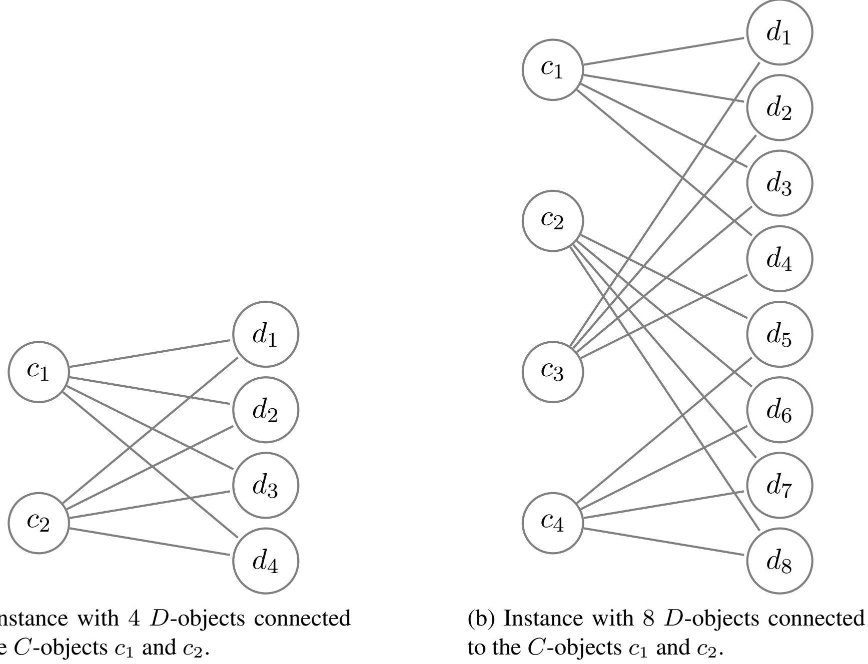
Figure 4.1: Model of two classes.

So far we have assumed that x is the total number of objects of class C. As we might as well be only interested in the number of D-objects connected to a subset of all existing C-objects (i.e. x is the number of a subset of all C-objects), we need to adapt the upper bound on the number y of connected objects. In this case the upper bound is different, because the D-objects may additionally be connected to other objects of class C. The following example illustrates this.

Example 4.2. Reconsider the model in Figure 4.1. Figure 4.2b shows an instance with two additional objects of class C. We are still only interested in objects connected to c_1 and c_2. In this instance we can connect 8 D-objects to these two C-objects. Hence, we get $y = 8$ for $x = 2$, if the instance consists of more than x objects of class C. This value exceeds the upper bound for y from (4.5).

Proposition 4.3. *Given an association $C\ {}_{m..M}{\Rightarrow}_{n..N}\ D$. Let $|C|$ be the number of C-objects and $|D|$ the number of D-objects. Furthermore, let $O \subseteq \mathrm{obj}(C)$ be a set of some C-objects, and let $r(O) := \{\, p \mid o \in O, (o,p) \in r \,\} = \bigcup_{o \in O} \gamma_r(o)$ be the set of related D-objects, where $x = |O|$ is the number of objects in O and $y = |r(O)|$ the number of related objects ($1 \leq x \leq |C|$ and $1 \leq y \leq |D|$). Then y is bounded by*

$$\max(\left\lceil \frac{nx}{M} \right\rceil, n \cdot \mathrm{sgn}(x)) \leq y \leq Nx \quad (4.6)$$



(a) Instance with 4 D-objects connected to the C-objects c_1 and c_2.

(b) Instance with 8 D-objects connected to the C-objects c_1 and c_2.

Figure 4.2: 2 instances of the model in Figure 4.1.

Proof. Regarding the upper bound we observe that a single C-object is linked to at most N objects of class D. In the maximal case, the D-objects linked to distinct C-objects are pairwise different, hence y is bounded from above by $N \cdot |O|$. As x is only a subset of all existing C-objects, the D-objects can be additionally linked to C-objects that are not in O and hence we are not limited by the requirements of the D-objects (which need at least m objects of class C).

The lower bound follows directly from the inequalities (4.1) and (4.2). □

The upper bound Nx is tight, i.e. the value can actually be reached by some satisfying instance of the association (see Corollary 4.6).

Characterisation of solutions

We now characterise the solutions of the inequalities by deriving restrictions for the values of x and y that yield a solution to the inequalities.

The inequalities (4.1) are equivalent to (4.4); hence y exists if x satisfies

$$\lfloor \frac{Nx}{m} \rfloor \geq \lceil \frac{nx}{M} \rceil \ .$$

Since $\lfloor \frac{nx}{M} + 1 \rfloor \geq \lceil \frac{nx}{M} \rceil$, rewriting this as

$$\lfloor \frac{Nx}{m} \rfloor \geq \lfloor \frac{nx}{M} + 1 \rfloor$$

still yields a sufficient condition for the existence of y. Omitting $\lfloor\ \rfloor$ yields $\frac{Nx}{m} \geq \frac{nx}{M} + 1$, which can be rewritten as

$$x \geq \frac{mM}{MN - mn} \quad .$$

We conclude that y exists and we can hence find a solution to the inequalities for an arbitrary number x of C-objects if $m < M$ or $n < N$, provided that x is large enough. This is formally stated in the following lemma.

Lemma 4.4. *Let $m \neq M$ or $n \neq N$. Then the inequalities* (4.1) *are solvable for every $x \geq \frac{mM}{MN-mn}$ and for every $y \geq \frac{nN}{MN-mn}$.*

Note that in cases where both of the intervals have identical values for the lower and the upper bound, we might not find a solution for every value x as stated in Proposition 4.4.

The next lemma shows that we can still find a solution for some values x, if $m = M$ and $n = N$. Furthermore, we can find a solution for some multiple of any value x.

Lemma 4.5. *Let $m = M$ and $n = N$, and let $g = \gcd(m, n)$.*[2] *Then x, y is a solution of the inequalities* (4.1) *if and only if $x = \frac{m}{g}i$ and $y = \frac{n}{g}i$ for some $i > 0$.*

Proof. For $m = M$ and $n = N$, the inequalities (4.1) are equivalent to $\frac{n}{g}x = \frac{m}{g}y$. Since the coefficients are co-prime, x and y have to be of the form $x = \frac{m}{g}i$ and $y = \frac{n}{g}i$ for some $i > 0$. Obviously each of these numbers is a solution. □

Suppose we have z objects of class D. The following corollary allows us to conclude that it is always possible to extend the given D-objects to a full instance by adding a suitable number of C- and D-objects.

Corollary 4.6. *Let $C\ _{m..M}{\Rightarrow}_{n..N}\ D$ be an association such that C and D are distinct, and let $z > 0$. Then, for some number d, there exists an instance with dz objects of class D satisfying the association.*

Proof. If $m \neq M$ or $n \neq N$, then the inequalities corresponding to the association are solvable for every $y \geq \frac{nN}{MN-mn}$ (Lemma 4.4); hence we may choose e.g. $d = \lceil \frac{nN}{(MN-mn)z} \rceil$. For $m = M$ and $n = N$ the solutions for y are of the form $\frac{n}{g}i$ (Lemma 4.5); hence we may choose e.g. $d = \frac{n}{g}$. □

Special Cases

An interesting case is the one where in a collection of C-objects one C-object is only connected to the minimally required number n of D-objects via an association u of type $C\ _{m..M}{\Rightarrow}_{n..N}$ D. We want to know the maximal number of D-objects connected to all C-objects under this condition. We will need this result to calculate the reduced lower bounds of multiplicities in Chapter 5.

The number ℓ of links instantiating the association is bounded by

[2] The function $\gcd(m, n)$ returns the greatest common divisor of m and n.

$$nx \leq \ell \leq N \cdot (x-1) + n \tag{4.7}$$
$$my \leq \ell \leq My \tag{4.8}$$

which leads to the inequality

$$my \leq Nx + (n - N) \tag{4.9}$$

Hence we obtain the following result.

Proposition 4.7. *Let $C\ {}_{m..M}\!\Rightarrow_{n..N}\ D$ be an association and let $x \geq 1$ be the number of C-objects. Then the number y of D-objects connected to the x C-objects is bounded by*

$$y \leq \left\lfloor \frac{Nx + (n - N)}{m} \right\rfloor \tag{4.10}$$

if one C-object is required to be connected to exactly n objects of class D.

In a similar manner we determine the minimal number of D-objects connected to a given number of C-objects via an association u of type $C\ {}_{m..M}\!\Rightarrow_{n..N}\ D$, if one of those C-objects is connected to the maximal possible number of D-objects (which is N). This result will be used in Chapter 5 to calculate the reduced upper bounds of multiplicities.

The ℓ links instantiating this association are bounded by

$$n(x-1) + N \leq \ell \leq Nx \tag{4.11}$$
$$my \leq \ell \leq My \tag{4.12}$$

which leads to the inequality

$$n(x-1) + N \leq My \tag{4.13}$$

Moreover we want one C-object to actually use the upper bound of N, hence we require $y \geq N$.

We obtain the following result.

Proposition 4.8. *Given an association $C\ {}_{m..M}\!\Rightarrow_{n..N}\ D$. Let x' be the number of objects of class C and x be the number of a specific subset of C-objects with $1 \leq x \leq x'$. Then the number y of D-objects connected to the x C-objects is bounded from below by*

$$\max(\lceil \frac{nx + (N - n)}{M} \rceil, N) \leq y \tag{4.14}$$

if one of those C-objects is connected to exactly N objects of class D.

4.2 Composition of associations

The concept of equations over association chains that we will deal with in Chapter 5 is based on the composition of association chains. This section investigates basic definitions and properties of relations instantiating such association chains.

Definition 4.9 (Association chain)**.** An *association chain* is a sequence of associations u_i of type $C_{i-1}\ {}_{m_i..M_i}\!\!\Rightarrow_{n_i..N_i}\ C_i$ (for $i = 1,\dots,k$) such that the classes C_i are distinct from each other (see Figure 4.3).

Figure 4.3: Association chain consisting of k associations.

Definition 4.10 (Composition of relations)**.** The composition of two relations r_u, r_v (instantiating an association chain uv) is defined as $r_u \circ r_v = \{\, (o,p) \mid (o,q) \in r_u,\, (q,p) \in r_v \,\}$.

In other words, the composition of two relations (the first between objects of classes C_0 and C_1 and the second between objects of classes C_1 and C_2) can be defined as pairs of objects (o,p) ($o \in C_0, p \in C_2$) that are connected via objects of the intermediate class C_1.

Let $\Delta_{n,M}(x)$ denote the expression $\max(\lceil \frac{n \cdot x}{M} \rceil,\ n \cdot \mathrm{sgn}(x))$, which gives the minimal number of D-objects that x objects of class C have to be linked to in a satisfying instance. We can now compute the type of a composed association as stated in the following proposition.

Proposition 4.11 (Composition of association types)**.** *Let r_i be a relation of type $C_{i-1}\ {}_{m_i..M_i}\!\!\Rightarrow_{n_i..N_i}$ C_i for $i = 1,\dots,k$. Then the composition $r_1 \circ \dots \circ r_k$ is of type*

$$C_0\ {}_{\mu_k..\prod_{i=1}^{k} M_i}\!\!\Rightarrow_{\nu_k..\prod_{i=1}^{k} N_i}\ C_k\ ,$$

where the lower bounds are defined by the recursions $\mu_0 = \nu_0 = 1$, $\mu_i = \Delta_{m_{k-i+1},N_{k-i+1}}(\mu_{i-1})$, and $\nu_i = \Delta_{n_i,M_i}(\nu_{i-1})$, for $i = 1,\dots,k$.

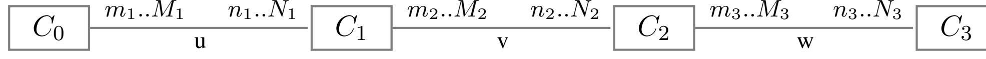
Proof. Regarding the upper bounds, we see that applying the upper bound from Proposition 4.3 (i.e. the upper bound for the number of connected objects) repeatedly for one association after the other within the association chain (starting from C_0) yields the maximal number of partner objects in each step. After k steps, we get the composed upper bound $\prod_{i=1}^{k} N_i$.

Regarding the lower bound, like for the upper bound, we apply the lower bound from Proposition 4.3 repeatedly (in total k times), yielding the minimal number of partner objects ν_k.

The inverse direction regarding the multiplicities $\nu_k..\prod_{i=1}^{k} M_i$ is symmetric.

□

Figure 4.4 illustrates the composition of associations. It shows a chain of three associations viewed as one composed association.



(a) Three associations ...

(b) ... viewed as one.

Figure 4.4: Composition of three associations, with $\mu_3 = \Delta_{m_1,N_1}(\mu_2)$, and $\nu_3 = \Delta_{n_3,M_3}(\nu_2)$.

Example 4.12. Consider the association chain uvw depicted in Figure 4.5. The composed type $C_0\ {}_{a..A}{\Rightarrow}_{b..B}\ C_3$ is calculated as follows:

For the upper bounds we get

$$A = \prod_{i=1}^{3} M_i = 1 \cdot 1 \cdot 2 = 2 \quad \text{and} \quad B = \prod_{i=1}^{3} N_i = 3 \cdot 4 \cdot 3 = 36 \ .$$

The lower bounds are

$$\begin{aligned} a = \mu_3 = \Delta_{m_1,N_1}(\Delta_{m_2,N_2}(\Delta_{m_3,N_3}(1))) = \\ \Delta_{m_1,N_1}(\Delta_{m_2,N_2}(\max(\lceil \tfrac{m_3 \cdot 1}{N_3} \rceil,\ m_3 \cdot \operatorname{sgn}(1)))) = \\ \Delta_{m_1,N_1}(\Delta_{m_2,N_2}(\max(\lceil \tfrac{1 \cdot 1}{3} \rceil,\ 1))) = \Delta_{m_1,N_1}(\Delta_{m_2,N_2}(1)) = \\ \Delta_{m_1,N_1}(\max(\lceil \tfrac{1 \cdot 1}{4} \rceil,\ 1)) = \Delta_{m_1,N_1}(1) = \\ \max(\lceil \tfrac{1 \cdot 1}{3} \rceil,\ 1) = 1 \end{aligned}$$

and

$$\begin{aligned} b = \nu_i = \Delta_{n_3,M_3}(\Delta_{n_2,M_2}(\Delta_{n_1,M_1}(1))) = \\ \Delta_{n_3,M_3}(\Delta_{n_2,M_2}(\max(\lceil \tfrac{n_1 \cdot 1}{M_1} \rceil,\ n_1 \cdot \operatorname{sgn}(1)))) = \\ \Delta_{n_3,M_3}(\Delta_{n_2,M_2}(\max(\lceil \tfrac{1 \cdot 1}{1} \rceil,\ 1))) = \Delta_{n_3,M_3}(\Delta_{n_2,M_2}(1)) = \\ \Delta_{n_3,M_3}(\max(\lceil \tfrac{2 \cdot 1}{1} \rceil,\ 2)) = \Delta_{n_3,M_3}(2) = \\ \max(\lceil \tfrac{2 \cdot 2}{2} \rceil,\ 2) = 2 \ . \end{aligned}$$

Hence the composed association $u \circ v \circ w$ is of type $C_0\ {}_{1..2}{\Rightarrow}_{2..36}\ C_3$.

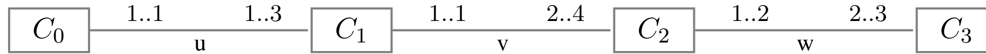


Figure 4.5: Association chain with three associations.

Properties of composed associations

We now describe several properties of association chains of the following form:

$$C_0 \;{}_{m_1..M_1}\!\Rightarrow_{n_1..N_1} \cdots {}_{m_k..M_k}\!\Rightarrow_{n_k..N_k} C_k$$

The first observation is that we can build a satisfying instance of the association chain containing a multiple of an arbitrary number of objects of any class within the association chain.

Proposition 4.13. *For every number $z > 0$ and $i = 0, \ldots, k$ there is a number d, such that there exists an instance with dz objects of class C_i satisfying the association chain.*

Proof. Starting from C_i, apply corollary 4.6 in both directions (in total k times). □

This insight allows us to state that an instance of an association chain, where all objects of one class (i.e. of one side of an association within the chain) are satisfied, can always be extended to a satisfying instance of the whole chain.

Proposition 4.14. *Let $\mathfrak{I}$ be an instance of the association chain such that each C_i-object is linked to at least n_{i+1} and at most N_{i+1} different C_{i+1}-objects. Then $\mathfrak{I}$ can be extended to a satisfying instance, $\mathfrak{I}^*$, of the chain.*

Proof. If each C_{i+1}-object is linked to at least m_i and at most M_i different C_i-objects, $\mathfrak{I}$ is already a satisfying instance and we are done. Otherwise we have to augment $\mathfrak{I}$ by further objects and links to reach this situation.

Let ε_{i+1} be the number of additional links that are required to link every C_{i+1}-object to at least m_{i+1} C_i-objects. Consider the union of $n = \prod_{i=1}^{k} n_i$ renamed copies of instance $\mathfrak{I}$. In total, the C_{i+1}-objects now need $n \cdot \varepsilon_{i+1}$ additional links. Therefore we add $x = \frac{n}{n_{i+1}} \cdot \varepsilon_{i+1}$ further C_i-objects and link each of them to n_{i+1} different C_{i+1}-objects. Let $\mathfrak{I}'$ be the instance consisting of the copies of $\mathfrak{I}$ and the additional C_i-objects and links.

It remains to satisfy the needs of the additional C_i-objects regarding links to C_{i-1}-objects. By Proposition 4.13, there is a number d such that there is a satisfying instance $\mathfrak{I}''$ of the chain $C_0 \Rightarrow \cdots \Rightarrow C_i$ with dx objects of class C_i. Therefore we take d copies of $\mathfrak{I}'$ and add $\mathfrak{I}''$. Repeating this step for every i we obtain $\mathfrak{I}^*$. □

We can now show that the bounds stated in Proposition 4.11 are indeed tight. This means that there is a satisfying instance where at least one object of the first class in the chain is linked to as many objects of the last class as given by these bounds.

Proposition 4.15 (Tightness)**.** *Let $C_0 \;{}_{a..A}\!\Rightarrow_{b..B} C_k$ be the type obtained by composing a chain of associations according to Proposition 4.11. Then the intervals $a..A$ and $b..B$ are tight, i.e., for each of the interval bounds a, A, b and B there is an instance satisfying the association chain that contains a C_0-object (resp. C_k-object) linked to this number of C_k-objects (resp. C_0-objects).*

Proof. Given x objects of class C_i, it is always possible to link them to $\Delta_{n_{i+1},M_{i+1}}(x)$ objects of class C_{i+1} such that each C_i-object has exactly n_{i+1} partners, by distributing the links uniformly (see the discussion on balanced sequences in [21]). Iterating this step we see that a single C_0-object can be connected to $b = \nu_k$ C_k-objects. By Proposition 4.14 this partial instance can be extended to a satisfying instance of the chain. □

Although the bounds can be reached, it may happen that we can not construct an instance for every value of the interval formed by the bounds.

Proposition 4.16 (Gaps)**.** *Let C_0 $_{m..M}\Rightarrow_{n..N}$ C_k be the composition of a chain of k associations (as in Proposition 4.11). Then there may be values y, $n < y < N$, such that in no satisfying instance of the association chain any C_0-object is linked to exactly y objects of class C_k.*

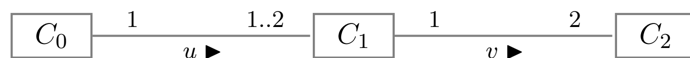
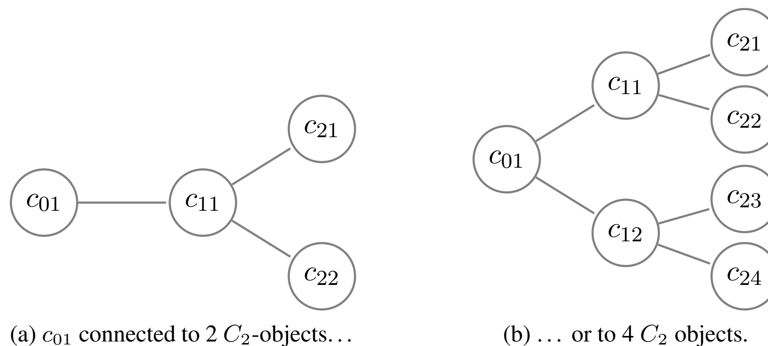


Figure 4.6: A chain of two associations admitting only satisfying instances with an even number of C_2-objects.

Proof. Consider the association chain in Figure 4.6. The composed association is of type C_0 $_{1..1}\Rightarrow_{2..4}$ C_2, but there does not exist a satisfying instance where some object of class C_0 is connected to exactly three objects of class C_2. Each C_0-object is connected to either two or four C_2-objects (see Figure 4.7). □



(a) c_{01} connected to 2 C_2-objects... (b) ... or to 4 C_2 objects.

Figure 4.7: Possible instances of the model in Figure 4.6 with 1 object of class C_0.

Since the composition of relations is associative, one might expect that this property transfers to the composition of associations. Maybe surprisingly at first sight, this is not the case.

Proposition 4.17. *The composition of associations is not associative.*

Proof. Consider the chain of three associations u, v, and w in Figure 4.8. Computing the composition uvw stepwise as $(uv)w$ we obtain the association type $C_0 \ _{1..2}\!\Rightarrow_{3..6} \ C_3$ (Figure 4.8(a)). Computing it as $u(vw)$ we obtain the type $C_0 \ _{1..2}\!\Rightarrow_{2..6} \ C_3$ (Figure 4.8(b)), which is different from the previous one. □

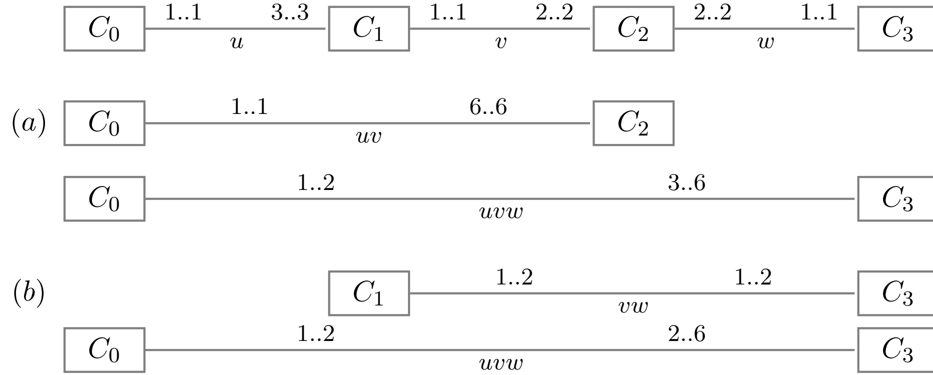


Figure 4.8: Chain illustrating the non-associativity of association composition.

In this particular example the reason for the different results is the subchain $C_1 \ _1\!\Rightarrow_2 \ C_2 \ _2\!\Rightarrow_1 \ C_3$; relations instantiating it have the composition type $C_1 \ _1\!\Rightarrow_1 \ C_3$ or $C_1 \ _2\!\Rightarrow_2 \ C_3$, but neither $_1\!\Rightarrow_2$ nor $_2\!\Rightarrow_1$. In the composed type $C_1 \ _{1..2}\!\Rightarrow_{1..2} \ C_3$ this information is lost, leading to bounds in $u(vw)$ that are not tight. In general none of the composition orders is guaranteed to yield tight lower bounds. Instead, the composition of a chain has to be done according to Proposition 4.11, with the right-hand multiplicity computed from left to right and the left-hand one from right to left.

5 Reducing Multiplicities and Models

Make everything as simple as possible, but not simpler.

Albert Einstein

In this chapter we explain why non-minimal multiplicities should be detected and how stricter bounds can be obtained.

5.1 Motivation

Reducing multiplicities means to find minimal multiplicities for associations. The problem of finding minimal multiplicities was identified in [16] as a relevant issue. In complex diagrams it may happen that the lower or upper bound of some multiplicities can never be reached due to other restrictions. One possible reason is that there are several associations or association chains between the same two classes. Thus, the parallel association chain may impose stronger restrictions on the relationship between the two classes than some specific association u. If we want all associations (or association chains) to hold simultaneously, we can hence reduce the multiplicities of association u without changing the meaning of the model.

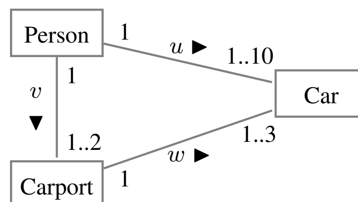


Figure 5.1: Model of persons, their cars and carports.

Consider again the Carport-example from Chapter 2 (see Figure 5.1) . The instance in Figure 5.2 is a satisfying instance of the model. Car c_2 belongs to person p_1, but is parked in carport cp_1, which belongs to person p_2. In general, this will be an acceptable instance, but we can imagine a different scenario as well. If we want to force persons to park their cars only in their own carports, we need to specify this by adding additional constraints. In this case, we need to equate the direct association Person-Car with the parallel association chain Person-Carport-Car to guarantee that the cars directly connected to a person are the same as those connected via

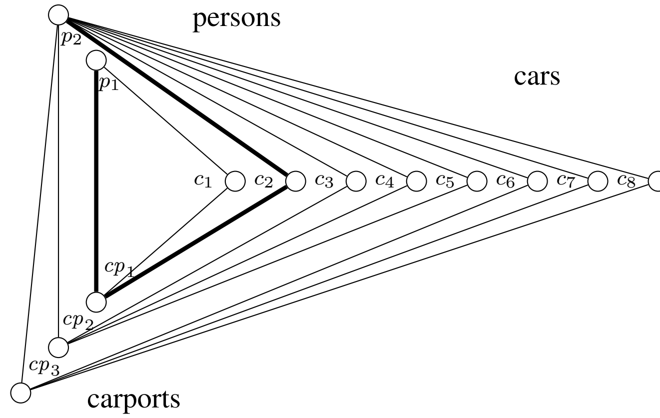


Figure 5.2: Instance where one car of person 2 is parked in his neighbour's carport.

the relation instantiating the association chain. By adding this constraint, we can observe that a person may now possess at most 6 cars, as each person may only have 2 carports, with 3 parking slots each.

Detecting and eliminating such unreachable bounds provides valuable feedback to the user, as they may hint at some underlying misconception. In configuration management, where we deal with physical objects, reducing multiplicities can also save money. Suppose you uncover that due to some constraint it is not possible to park 3 cars in any carport, but at most 2. Then it suffices to build smaller, cheaper carports that only have 2 parking slots. Last, but not least minimising multiplicities is even a means to detect that a model is not E-satisfiable (i.e. that there does not exist any instance satisfying a given equation E).

5.2 Equations over association chains

To reduce multiplicities we have to extend the definition of models by a third component, namely a set $\mathcal{E}$ of equations over association chains: $\mathfrak{M} = (\mathcal{C}, \mathcal{A}, \mathcal{E})$.

An *equation* over $\mathcal{A}$ is of the form $u_1 \cdots u_k = v_1 \cdots v_l$, where each u_i and v_j is of the form a or a^{-1} for some association a. For an association a of type $C \;_{m..M}\!\Rightarrow_{n..N}\; D$ the *inverse association* a^{-1} has type $D \;_{n..N}\!\Rightarrow_{m..M}\; C$. In UML this corresponds to navigating along the association in the opposite direction.

Example 5.1. We extend the formalisation of the model in Figure 5.1 by a set of equations. Hence we have $\mathfrak{M} = (\mathcal{C}, \mathcal{A}, \mathcal{E})$, where

$$\begin{aligned}
\mathcal{C} &= \{\text{Car}, \text{Carport}, \text{Person}\} \;, \\
\mathcal{A} &= \{\, u\colon \text{Person} \;_{1..1}\!\Rightarrow_{1..10} \text{Car}, \\
&\qquad v\colon \text{Person} \;_{1..1}\!\Rightarrow_{1..2} \text{Carport}, \\
&\qquad w\colon \text{Carport} \;_{1..1}\!\Rightarrow_{1..3} \text{Car} \,\} \;, \\
\mathcal{E} &= \{u = vw\} \;.
\end{aligned}$$

Equations over association chains are constraints used to specify that objects reachable via one association (chain) are the same that are reachable via some parallel association (chain). Hence, the modeller can (and even has to) make his implicit assumptions explicit by adding equality constraints.

Definition 5.2 (E-satisfying instance)**.** An instance $\mathfrak{I}$ *E-satisfies* a model $\mathfrak{M}$ if $\mathfrak{I}$ satisfies $\mathfrak{M}$ and if all equations in $\mathcal{E}$ are satisfied, i.e., for each $u_1 \cdots u_k = v_1 \cdots v_l \in \mathcal{E}$ the composed relations $\mathrm{rel}(u_1) \circ \cdots \circ \mathrm{rel}(u_k)$ and $\mathrm{rel}(v_1) \circ \cdots \circ \mathrm{rel}(v_l)$ are equal (as sets).

Example 5.3. According to Example 2.11, Figure 5.2 is a satisfying instance of the model in Figure 5.1. It is not E-satisfying, however, if we want the equation $(u = vw) \in \mathcal{E}$ to hold: The composed relation $\mathrm{rel}(v) \circ \mathrm{rel}(w)$ contains the pair (p_1, c_2), which does not occur in the relation $\mathrm{rel}(u)$. The links violating the equation are highlighted in Figure 5.2. By deleting the object c_2 and the corresponding links we obtain an E-satisfying instance.

Definition 5.4 (E-satisfiable model)**.** A model $\mathfrak{M} = (\mathcal{C}, \mathcal{A}, \mathcal{E})$ is *E-satisfiable*, if there exists an instance that satisfies the model and respects all equality constraints specified in the set of equations $\mathcal{E}$.

Example 5.5. The model in Figure 5.3 is not E-satisfiable with $\mathcal{E} = \{u = v\}$, as each C-object has to be connected to 3 objects of class D via association u and to one or two D-objects via association v. Hence, we cannot find a relation instantiating both associations at the same time.

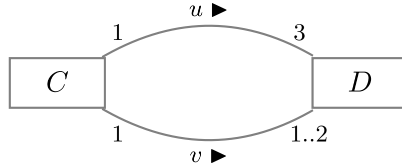


Figure 5.3: Two associations u and v between the same classes allowing no E-satisfying instance for $\mathcal{E} = \{u = v\}$.

The concept of equating associations or association chains can as well be expressed in the *Object Constraint Language (OCL)* of UML.

Example 5.6. The constraint $u = vw$ from Example 5.1 can be expressed in OCL as follows:

```
context Person:

  inv: self->collect(p: Person | p.car)->asSet()
     = self->collect(p: Person | p.carport)
           ->collect(c: Carport| c.car)->flatten()->asSet()
```

As this notation can become very cumbersome, we express the constraints in the more intuitive and simple notation of equations.

5.3 Reducing multiplicities

Reducing multiplicities is a three-step approach. First, we need to define some equation(s) as part of the model. Then we calculate the composed type of the association chains for both sides of each equation (as described in Section 4.2). Finally, we can reduce the multiplicities of associations within the association chains.

The goal is to exploit the information encoded in equations like $u = vw$ from Example 5.1 to tighten multiplicity bounds. In general we obtain two different composed association types $C_0 \;_{m..M}{\Rightarrow}_{n..N}\; C_k$ and $C_0 \;_{m'..M'}{\Rightarrow}_{n'..N'}\; C_k$, one for each side of the equation, which by the semantics of equations both characterise the relation instantiating the chains. Obviously a relation satisfies both types if it satisfies the intersection type, as formally stated in Lemma 2.12. We call the intersection type *objective type* and the corresponding interval *objective interval*. If the intersection is empty (i.e. $[\max(n, n')..\min(N, N')]$ or $[\max(m, m')..\min(M, M')]$ leads to an empty interval), then the equation is unsatisfiable and the model is not E-satisfiable. An interval $a..A$ is *empty*, if $a > A$. Two intervals $a..A$ and $b..B$ *overlap*, if their intersection $[\max(a, b)..\min(A, B)]$ is not empty.

Lemma 5.7. *Given an equation $U = V$ of two association chains between classes C_0 and C_k, let $C_0 \;_{a_u..A_u}{\Rightarrow}_{b_u..B_u}\; C_k$ and $C_0 \;_{a_v..A_v}{\Rightarrow}_{b_v..B_v}\; C_k$ be the type of the composition of the chains U and V, respectively. Then the resulting intervals $a_u..A_u$ and $a_v..A_v$ have to be overlapping (as well as $b_u..B_u$ and $b_v..B_v$) for the model to be E-satisfiable.*

Bounds from the objective interval (like $\min(M, M')$ and $\min(N, N')$) obtained for the whole chain can now be propagated to the individual associations within the association chains to tighten the individual multiplicities. For the following sections, consider an association chain of the form $C_0 \;_{m_1..M_1}{\Rightarrow}_{n_1..N_1} \cdots \;_{m_k..M_k}{\Rightarrow}_{n_k..N_k}\; C_k$ and an objective type $C_0 \;_{a..A}{\Rightarrow}_{b..B}\; C_k$.

Lower Bounds

First, we investigate how to tighten the lower bounds of multiplicities within a chain of associations. This means that we increase all lower bounds of an association chain that are not reachable.

Basically we may increase the lower bounds of multiplicities as long as the composed bound is smaller than b for all combinations of admissible relations. Suppose we want to find a tighter bound $n'_i \geq n_i$ for association u_i (which is of type $C_{i-1} \;_{m_i..M_i}{\Rightarrow}_{n_i..N_i}\; C_i$). From our objective interval we know that we need to reach at least b objects of class C_k. We choose the upper bound N_j for all other associations u_j, which means that each C_{j-1}-object is linked to the maximal possible number of objects of class C_j. Furthermore we take the upper bound for all objects of class C_{i-1} but one. This single object shall be connected to the least possible objects of class C_i. Hence we have to adapt the current lower bound n_i, if it is still not possible to reach the composed lower bound of b. The new lower bound n'_i is therefore obtained by taking the smallest value α for this single C_{i-1}-object such that the composed bound does not fall below b.

We use the result of Proposition 4.3 to maximise the connected objects for all associations $u_j,\ j \neq i$, as we only deal with a subset of all existing objects of each class. To maximise the connected objects of association u_i we use the result of Proposition 4.7. This proposition gives

us the upper bound of all C_i-objects connected to a specified number of objects of class C_{i-1}, if one of the C_{i-1}-objects is connected to the minimally required objects.

Proposition 4.7 assumes that there is a total of x objects of class C (corresponding to class C_{i-1} here). Therefore, we need to satisfy all D-objects (corresponding to the C_i-objects) using these C_{i-1}-objects. This is too restrictive in this scenario, because x is only the number of C_{i-1}-objects connected to one specific C_0-object. If there are more objects of class C_0, we can get more than x objects of class C_{i-1} in total. Nevertheless we are only interested in the C_i-objects connected to the x objects of class C_{i-1}. We therefore need to adapt the formula, resulting in the following proposition. It gives a recursive formula for the calculation of a tighter lower bound for relation r_i.

Proposition 5.8. *Let r_i be a relation of type $C_{i-1}\ {}_{m_i..M_i}\!\!\Rightarrow_{n_i..N_i} C_i$ for $i = 1, \dots, k$. Suppose $r_1 \circ \cdots \circ r_k$ is known to be of type $C_0 \Rightarrow_{b..B} C_k$, i.e., each object of class C_0 is known to be related to at least b and at most B objects of class C_k. Then each relation r_i is also of type $C_{i-1} \Rightarrow_{n'_i..N_i} C_i$, where $n'_i = \min\{\, \alpha \geq n_i \mid f_k(i,\alpha) \geq b \,\}$, and f_k is defined recursively as*

$$f_0(i,\alpha) = 1 \tag{5.1}$$

$$f_j(i,\alpha) = \begin{cases} N_j \cdot f_{j-1}(i,\alpha) & \text{for } j \neq i \\ N_j \cdot (f_{j-1}(i,\alpha) - 1) + \alpha & \text{for } j = i \end{cases} \tag{5.2}$$

.

To calculate the new value of the lower bound, n'_i, of association u_i, we start with one object of class C_0, i.e. $f_0(i,\alpha) = 1$ (formula (5.1)). We iteratively calculate $f_j(i,\alpha)$ by formula (5.2) for $j = 1 \dots k$. In each iteration we proceed to the next association by increasing j. We calculate the required partner-objects (of class C_j) for all C_{j-1}-objects (the number of C_{j-1}-objects corresponds to the value calculated in the previous step, i.e. $f_{j-1}(i,\alpha)$).

By rewriting the recursive formula for $f_k(i,\alpha)$ from Proposition 5.8, we get an explicit formula for the lower bound. First we combine the calculation for association u_i with the calculation for all associations u_j with $j > i$:

$$(N_i \cdot (f_{i-1}(i,\alpha) - 1) + \alpha) \cdot \prod_{j=i+1}^{k} N_j \;=\; f_k(i,\alpha) \;\geq\; b \;.$$

Then we include all associations u_j with $j < i$ by replacing $f_{j-1}(i,\alpha)$:

$$(N_i \cdot (\prod_{j=1}^{i-1} N_j - 1) + \alpha) \cdot \prod_{j=i+1}^{k} N_j \;=\; f_k(i,\alpha) \;\geq\; b \;,$$

which is the same as

$$\prod_{j=1}^{k} N_j - \prod_{j=i}^{k} N_j + \alpha \cdot \prod_{j=i+1}^{k} N_j \;\geq\; b$$

and hence

$$\alpha \;\geq\; \frac{b + \prod_{j=i}^{k} N_j - \prod_{j=1}^{k} N_j}{\prod_{j=i+1}^{k} N_j} \;.$$

The reduced lower bound of association u_i, n'_i, can thus be calculated using the explicit formula

$$n'_i = \max\left(\left\lceil \frac{b}{\prod_{j=i+1}^{k} N_j} + N_i - \prod_{j=1}^{i} N_j \right\rceil, n_i\right) \quad . \tag{5.3}$$

Upper Bounds

Now we want to investigate the tightening of the upper bounds. We want to decrease all upper bounds of multiplicities within an association chain that are not reachable.

Similar to the lower bounds, we may reduce the upper bounds of multiplicities of single associations as long as the composed upper bound is greater than B for all combinations of admissible relations. Suppose we want to find a tighter bound $N'_i \leq N_i$ for relation r_i. We assume the lower bounds for all other relations r_j, which corresponds to linking each C_{j-1}-object to the minimal possible number of objects of class C_j. From (4.5) we know that the lower bound on the connected objects for relation r_j is Δ_{n_j,M_j}. We choose $\max(\lceil \frac{nx+(\alpha-n)}{M} \rceil, N)$ partner objects of class C_i for x objects of C_{i-1}, which corresponds to the bound given in Proposition 4.8. The new upper bound N'_i is obtained by taking the biggest value α such that the composed bound does not exceed B.

Tightening of intervals

The following proposition summarises the insights of the previous subsections and gives formulas to calculate reduced upper and lower bounds for the multiplicities of a relation r_i.

Proposition 5.9 (Reduction of multiplicities)**.** *Let r_i be a relation of type $C_{i-1}\ {}_{m_i..M_i}\!\Rightarrow_{n_i..N_i} C_i$ for $i = 1, \ldots, k$. Suppose $r_1 \circ \cdots \circ r_k$ is known to be of type $C_0\ {}_{a..A}\!\Rightarrow_{b..B} C_k$. Then each relation r_i is of type $C_{i-1}\ {}_{m'_i..M'_i}\!\Rightarrow_{n'_i..N'_i} C_i$ with the following new multiplicities:*

$$n'_i = \max(\lceil \tfrac{b}{\prod_{j=i+1}^{k} N_j} + N_i - \prod_{j=1}^{i} N_j \rceil, n_i) \tag{5.4}$$

$$N'_i = \begin{cases} N_i & \text{if } n_j = 0 \text{ for some } j = 1, \ldots, k \\ \max\{\, \alpha \leq N_i \mid g_k(i,\alpha) \leq B \,\} & \text{otherwise} \end{cases} \tag{5.5}$$

where g_k is defined recursively as

$$g_0(i,\alpha) = 1 \tag{5.6}$$

$$g_j(i,\alpha) = \begin{cases} \max(n_j, \lceil \frac{n_j \cdot g_{j-1}(i,\alpha)}{M_j} \rceil) & \text{for } j \neq i \\ \max(\lceil \frac{n_j \cdot (g_{j-1}(i,\alpha)-1)+\alpha}{M_j} \rceil, \alpha) & \text{for } j = i \end{cases} \tag{5.7}$$

The new multiplicities $m'_i..M'_i$ are defined symmetrically.

Proof. First, note that even though N'_i is defined as a maximum, it is in fact *smaller* than or equal to N_i. Likewise, n'_i is *larger* than or equal to n_i. Therefore $n'_i..N'_i$ potentially is a tighter multiplicity than $n_i..N_i$.

Second, observe that $f_k(i,\alpha)$ essentially is the product of the upper bounds N_j, with the only exception that instead of N_i the potential *lower* bound Δ_{α,M_i} is used for one of the C_{i-1}-objects. The explicit formula for n'_i is obtained by solving the recursion given in Proposition 5.8.

Likewise, $g_k(i,\alpha)$ is the composition of the lower bounds Δ_{n_j,M_j}, with the only exception that instead of Δ_{α,M_i} the potential *upper* bound α is used for one C_{i-1}-object. Formula (5.7) follows directly from (4.5) and Proposition 4.8.

□

To calculate $g_k(i,\alpha)$ we need to start from $\alpha = N_i$ and proceed downwards, until we find the first α such that g_k falls below B. This poses a problem for large values of N_i.

Note that, unlike for the lower bound, it is not possible to derive an explicit formula for reducing the upper bound of an association, due to rounding. By solving the recursive formula for g_k approximately, we obtain an estimate for N'_i. This value $N^\star_i$ can be derived from $g_k(i,\alpha)$ as follows:

$$\frac{n_i \cdot (g_{i-1}(i,\alpha) - 1) + \alpha)}{M_i} \cdot \prod_{j=i+1}^{k} \frac{n_j}{M_j} \;=\; g_k(i,\alpha) \;\leq\; B \;.$$

If we replace $g_{i-1}(i,\alpha)$ by its approximation, $\prod_{j=1}^{i-1} \frac{n_j}{M_j} \leq g_{i-1}(i,\alpha)$, we still know that the resulting expression is less or equal to $g_i(i,\alpha)$:

$$\frac{n_i \cdot (\prod_{j=1}^{i-1} \frac{n_j}{M_j} - 1) + \alpha)}{M_i} \cdot \prod_{j=i+1}^{k} \frac{n_j}{M_j} \;\leq\; g_k(i,\alpha) \;\leq\; B \;,$$

which can be rewritten as

$$\begin{aligned} \frac{\left(\frac{\prod_{j=1}^{i} n_j - n_i \cdot \prod_{j=1}^{i-1} M_j}{\prod_{j=1}^{i-1} M_j} + \alpha\right) \cdot \prod_{j=i+1}^{k} n_j}{\prod_{j=i}^{k} M_j} &= \\ \frac{\prod_{j=1}^{k} n_j - \prod_{j=1}^{i-1} M_j \cdot \prod_{j=i}^{k} n_j + \alpha \cdot \prod_{j=1}^{i-1} M_j \cdot \prod_{j=i+1}^{k} n_j}{\prod_{j=1}^{k} M_j} &\leq\; B. \end{aligned}$$

Hence, we can calculate $N^\star_i$ by the following formula:

$$
\begin{aligned}
N_i^\star &= \left\lfloor \frac{B \cdot \prod_{j=1}^{k} M_j + \prod_{j=1}^{i-1} M_j \cdot \prod_{j=i}^{k} n_j - \prod_{j=1}^{k} n_j + \alpha \cdot \prod_{j=1}^{i-1} M_j \cdot \prod_{j=i+1}^{k} n_j}{\prod_{j=1}^{i-1} M_j \cdot \prod_{j=i+1}^{k} n_j} \right\rfloor \\
&= \left\lfloor B \cdot M_i \cdot \prod_{j=i+1}^{k} \frac{M_j}{n_j} + n_i \cdot \left(1 - \prod_{j=1}^{i-1} \frac{n_j}{M_j} \right) \right\rfloor \quad .
\end{aligned}
$$

The minimum of $N_i^\star$ and N_i can be used as an initial value for α and hence the calculation can be done efficiently even for large N_i.

Reducing multiplicities of single associations (instead of multiplicities of associations within an association chain) is done by simply replacing the original interval by the stricter interval, i.e. by the intersection of both types.

Modified bounds of one association within the association chain may allow us to further reduce multiplicities of other associations of the chain. We therefore need to iterate the reduction of intervals given in Proposition 5.9, until we reach a fixed point. The algorithm in Figure 5.4 repeats this calculation until no association can be reduced any further.

1: **function** ChainReduction(U, T)
2: Let U be an association chain of the form $C_0 \;{}_{m_1..M_1}\!\!\Rightarrow_{n_1..N_1} \cdots {}_{m_k..M_k}\!\!\Rightarrow_{n_k..N_k} C_k$.
3: Let T be the association type $C_0 \;{}_{a..A}\!\!\Rightarrow_{b..B} C_k$.
4: $U_{\text{new}} \leftarrow U$
5: **repeat**
6: $U_{\text{old}} \leftarrow U_{\text{new}}$
7: Let U_{new} be obtained from U_{old} by reducing the multiplicities w.r.t. type T (Proposition 5.9).
8: **if** $m > M$ for some multiplicity $m..M$ in U_{new} **then**
9: **throw** exception "unsatisfiable"
10: **end if**
11: **until** $U_{\text{new}} = U_{\text{old}}$
12: **return** U_{new}
13: **end function**

Figure 5.4: Reducing the multiplicities of an association chain

Proposition 5.10 (Chain reduction). *Let $U = C_0 \;{}_{m_1..M_1}\!\!\Rightarrow \cdots \Rightarrow_{n_k..N_k} C_k$ be an association chain and T be the type $C_0 \;{}_{a..A}\!\!\Rightarrow_{b..B} C_k$. If* ChainReduction$(U, T)$ *in Figure 5.4 throws an exception, then there is no instance satisfying simultaneously the multiplicities of U and T. Otherwise* ChainReduction(U, T) *terminates and its result, U', has the property that the relations $r_1, \ldots, r_k$ instantiate U and their composition is of type T if and only if they instantiate U'.*

Proof. If the function throws an exception, some interval in the chain has been reduced to the empty interval in the attempt to guarantee that the composed relation is of type T. This implies that there is no non-trivial E-satisfying instance. Regarding termination, we observe that

the bounds of the new multiplicities in Proposition 5.9 are monotone with respect to the initial bounds. Iterating the bound computation never increases the intervals, therefore the loop either aborts with an exception or terminates with bounds that cannot be reduced any further. Proposition 5.9 guarantees that each single iteration of the loop preserves satisfiability with respect to U and T, therefore this property also holds on termination. □

Example 5.11. Consider the model in Figure 5.1 with the additional constraint $u = vw$. It contains the association Person-Car (u), which is of type Person $_{1..1}\Rightarrow_{1..10}$ Car.

According to Proposition 4.11 the composed upper bounds of the association chain vw are $A = \prod_{i=1}^{k} M_i = 1 \cdot 1 = 1$ and $B = \prod_{i=1}^{k} N_i = 2 \cdot 3 = 6$, and the composed lower bounds are

$$\begin{aligned} a = \mu_2 &= \Delta_{m_1,N_1}(\Delta_{m_2,N_2}(1)) \\ &= \Delta_{m_1,N_1}(\max(\lceil \tfrac{m_2 \cdot 1}{N_2} \rceil,\, m_2 \cdot \operatorname{sgn}(1))) \\ &= \Delta_{m_1,N_1}(\max(\lceil \tfrac{1 \cdot 1}{3} \rceil,\, 1 \cdot 1)) = \Delta_{m_1,N_1}(1) \\ &= \max(\lceil \tfrac{m_1 \cdot 1}{N_1} \rceil,\, m_1 \cdot \operatorname{sgn}(1)) \\ &= \max(\lceil \tfrac{1 \cdot 1}{2} \rceil,\, 1 \cdot \operatorname{sgn}(1)) = 1 \ \text{, and} \\ b = \nu_2 &= \Delta_{n_2,M_2}(\Delta_{n_1,M_1}(1)) = \Delta_{n_2,M_2}(1) = 1 \ . \end{aligned}$$

Hence, the composition uv is of type Person $_{1..1}\Rightarrow_{1..6}$ Car. From the equation $u = vw$ we know that the association u is also of this type.

As the interval of the composed association is the stricter one, we replace the original interval of u by the intersection of both intervals. After the reduction the association u is hence of type Person $_{1..1}\Rightarrow_{1..6}$ Car.

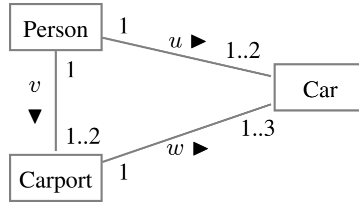


Figure 5.5: Model of persons, their cars and carports, where each person may possess at most two cars.

Example 5.12. Now consider the model in Figure 5.5 with the additional constraint $u = vw$. From Example 5.11 we know that the composition of the associations v and w is of type Person $_{1..1}\Rightarrow_{1..6}$ Car. According to equation $u = vw$ the composed association is also of type Person $_{1..1}\Rightarrow_{1..2}$ Car, which is the type of association u.

We now propagate the stricter upper bound to the associations within the chain with the formulas given in Proposition 5.9. First, we try to reduce the upper bound of association v (which is the first association within the chain): $N'_1 = \max\{\, \alpha \leq N_1 \mid g_2(1, \alpha) \leq 2 \,\}$. We

choose $\alpha = N_1 = 2$ and calculate $g_2(1, \alpha)$:

$$\begin{aligned}
g_2(1,2) &= \max(n_2, \lceil \frac{n_2 \cdot g_1(1,2)}{M_2} \rceil) \\
&= \max(1, \lceil \frac{1 \cdot \max(\lceil \frac{n_1 \cdot (g_0(1,2)-1)+2}{M_1} \rceil, 2)}{1} \rceil) \\
&= \max(1, (\max(\lceil \frac{1 \cdot (1-1)+2}{1} \rceil, 2))) \\
&= \max(1, (\max(2,2))) = \max(1,2) = 2 \enspace .
\end{aligned}$$

As $g_2(1, 2) \leq 2$, the current upper bound of association v can actually be reached and therefore be maintained.

Next, we try to reduce the upper bound of association w (which is the second association within the chain): $N_2' = \max\{ \alpha \leq N_2 \mid g_2(2, \alpha) \leq 2 \}$. We choose $\alpha = N_2 = 3$ and calculate $g_2(2, \alpha)$:

$$\begin{aligned}
g_2(2,3) &= \max(\lceil \frac{1 \cdot (g_1(2,3)-1)+3}{1} \rceil, 3) \\
&= \max(1 \cdot (\max(n_1, \lceil \frac{n_1 \cdot g_0(2,3)}{1} \rceil) - 1) + 3), 3) \\
&= \max((1 \cdot (\max(1, \lceil \frac{1 \cdot 1}{1} \rceil) - 1) + 3), 3) \\
&= \max((1 \cdot (1-1)+3), 3) = 3 \enspace .
\end{aligned}$$

As $g_2(2, 3) > 2$, we have to decrease α. Choosing $\alpha = 2$ results in $g_2(2, 2) = 2$ and hence $N_2' = 2$.

After the reduction the association w is of type Person $_{1..1}\Rightarrow_{1..2}$ Car.

5.4 Reducing Models

For a model $\mathfrak{M} = (\mathcal{C}, \mathcal{A}, \mathcal{E})$ we want to find some kind of normal form, such that all intervals range over values that can actually be reached in at least some E-satisfying instance. Although we cannot achieve this (yet), we can at least reduce the intervals (with algorithm CHAINREDUCTION), such that some unreachable values are eliminated.

Figure 5.6 specifies an algorithm for reducing the multiplicities of a model under an equation. It processes an equation E of the form $U = V$, computes the compositions for U and V, and calculates their intersection. This overlap is used to reduce the multiplicities of all associations within the chains U and V, until no further reductions are possible. If some multiplicity is reduced to the empty interval, the algorithm returns the exception “unsatisfiable” to signal that the model is not satisfiable with E. Otherwise the algorithm returns a new model consisting of the reduced associations.

Definition 5.13 (Model equivalence)**.** Two models $\mathfrak{M}$ and $\mathfrak{M}'$ are equivalent if they have the same E-satisfying instances.

```
1: function ModelReduction(𝔐, E)
2:     Let E be an equation of the form u_1 ··· u_k = v_1 ··· v_l.
3:     Let 𝔐 be a model containing the associations u_1, ..., u_k, v_1, ..., v_l.
4:     Let U and V be the association chains corresponding to u_1 ··· u_k and v_1 ··· v_l, respectively.
5:     U_new, V_new ← U, V
6:     repeat
7:         U_old, V_old ← U_new, V_new
8:         Let C a_1..A_1⇒b_1..B_1 D be the composition of U_old.
9:         Let C a_2..A_2⇒b_2..B_2 D be the composition of V_old.
10:        Let a..A = max(a_1, a_2)..min(A_1, A_2).
11:        Let b..B = max(b_1, b_2)..min(B_1, B_2).
12:        if a > A or b > B then
13:            throw exception "unsatisfiable"
14:        end if
15:        Let T be the type C a..A⇒b..B D.
16:        U_new ← ChainReduction(U_old, T)
17:        V_new ← ChainReduction(V_old, T)
18:    until U_new = U_old and V_new = V_old
19:    Let 𝔐_new be 𝔐 with the associations in U and V
       replaced by the reduced variants in U_new and V_new.
20:    return 𝔐_new
21: end function
```

Figure 5.6: Reducing a model $\mathfrak{M}$ with respect to an equation E

A model $\mathfrak{M}'$ obtained from $\mathfrak{M}$ by ModelReduction($\mathfrak{M}$, E) is equivalent to the original model $\mathfrak{M}$, as reducing multiplicities w.r.t. an equation $E \in \mathcal{E}$ does not influence the E-satisfying instances. Nevertheless, in general the satisfying instances of $\mathfrak{M}$ and $\mathfrak{M}'$ are not the same.

Proposition 5.14 (Model reduction). *Let $\mathfrak{M} = (\mathcal{C}, \mathcal{A}, \mathcal{E})$ be a model and $E \in \mathcal{E}$ be an equation. If* ModelReduction($\mathfrak{M}, E$) *(Figure 5.6) throws an exception then $\mathfrak{M}$ is not E-satisfiable. Otherwise* ModelReduction($\mathfrak{M}, E$) *terminates and its result $\mathfrak{M}_{new}$ is equivalent to $\mathfrak{M}$.*

Proof. If the function throws an exception, then some interval is empty after overlapping the intervals of the composition of U_{old} and the composition of V_{old}. This implies there is no non-trivial satisfying instance for both U and V at the same time.

Regarding termination, we distinguish two cases. In case of an exception the function aborts immediately. Otherwise the function ChainReduction is called which is guaranteed to terminate by Proposition 5.10. As the intervals $a..A$ and $b..B$ cannot grow by overlapping and since composition and ChainReduction are monotone in their arguments the loop stops once a fixed point is reached.

The result $\mathfrak{M}_{\text{new}}$ is equivalent to $\mathfrak{M}$ as intervals of associations in U and V are only modified due to calls of ChainReduction which is guaranteed to be equivalence preserving according

to Proposition 5.10. □

If a model has more than one equality constraint, the algorithm has to be run for each of them repeatedly (and alternating), until no more changes occur.

We call a model *reduced*, if all multiplicities are tightened by algorithm MODELREDUCTION.

Definition 5.15 (Reduced Model)**.** A model $\mathfrak{M} = (\mathcal{C}, \mathcal{A}, \mathcal{E})$ is *reduced w.r.t. an equation* $E \in \mathcal{E}$, if its multiplicities are reduced w.r.t. the equality constraint E (i.e. MODELREDUCTION$(\mathfrak{M}, E)$ has been applied to the model). We call a model *reduced*, if it has been reduced w.r.t. all equations in $\mathcal{E}$, i.e. MODELREDUCTION$(\mathfrak{M}, E)$ has been applied to the model for all $E \in \mathcal{E}$.

Reduced models are the basis for the observations and applications introduced in Chapter 6.

6 *Effects of Equality Constraints*

> Nobody really believes in equality anyway.
>
> Warren Farrell

In this chapter we discuss the effect of equations on models and their satisfying instances.

Checking satisfiability and computing minimal instances under equality constraints poses a hard challenge as we cannot easily reuse existing techniques. Enumerating instances in order to check the equations ("generate and test") is expensive or even infeasible (e.g. on infinite domains). To avoid this inefficient approach, we developed a more efficient technique that maps E-satisfiability to satisfiability and is applicable to many situations. For this to work, we first ensure that all intervals of multiplicities only range over values that can be realised in at least some E-satisfying instances.

6.1 Satisfiability under equality constraints

Adding equations may affect the satisfiability of a model. We start with some observations regarding the effect of equations.

A model $\mathfrak{M}$ may be satisfiable, but not E-satisfiable. A necessary condition for a model to be E-satisfiable with an equation $U = V$ is that the compositions of the chains U and V result in overlapping multiplicities.

The following example shows a model that is satisfiable, but not E-satisfiable, because the intervals on both sides of the equation E do not overlap.

Example 6.1. Consider the following model:

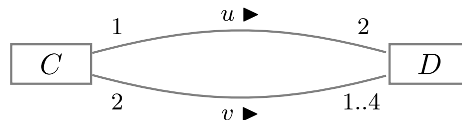


Figure 6.1 shows an instance satisfying $\mathfrak{M}$. Hence, the model is satisfiable. However, if we add the equality constraint $u = v$, then $\mathfrak{M}$ is not E-satisfiable: the association u requires one C-object for each D-object whereas v assigns two objects. The instance in Figure 6.1 is not E-satisfying the model, because c_1 is connected to all four D-objects via relation v, but only to

d_1 and d_2 via relation u, thus violating the equality $u = v$. c_2 violates the equality in a similar way. Equating both sides results in an empty interval which rules out any non-trivial E-satisfying instance.

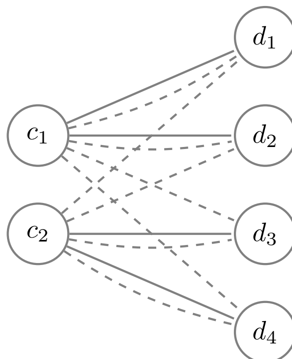
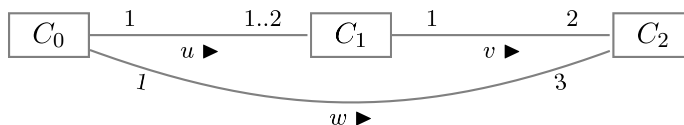


Figure 6.1: Satisfying instance of the model from Example 6.1: Black edges are instantiations of association u, dashed edges are instantiations of association v.

This behaviour occurs, if we need more objects for one association (chain) than are allowed for the parallel one. In this case the intervals on both sides of the equation do not overlap and we can not find any E-satisfying instances. However, overlapping intervals are not sufficient to guarantee E-satisfiability. As the following example illustrates, there are models that are satisfiable, but not E-satisfiable, although the multiplicities on both sides of the equation E overlap.

Example 6.2. Consider the following model $\mathfrak{M}$:



It is satisfiable as there exists a satisfying instance (Figure 6.2). If we add the equality constraint $uv = w$, then $\mathfrak{M}$ is no longer E-satisfiable, although the intervals of the composition uv, which is of type $C_0\ {}_{1..1}{\Rightarrow}_{2..4}\ C_2$ overlap with the intervals of association w (1..1 and 3..3). This happens, because the chain uv only allows for two or four C_2-objects for each single C_0-object whereas w assigns three objects. The composed association for chain $U = uv$ is of type $C_0\ {}_{1..1}{\Rightarrow}_{2..4}\ C_2$, the association $V = w$ is of type $C_0\ {}_{1..1}{\Rightarrow}_{3..3}\ C_2$. The intervals 1..1 and 2..4 overlap with 1..1 and 3..3, respectively, but the model is still not E-satisfiable.

Such situations can occur, because although the bounds of our composed associations are tight, there may still be values in between that are not realisable (see Proposition 4.16). If we try to reduce the multiplicities of equated association chains that have no values in common (i.e. the values allowed by one association chain correspond to the gaps of the other association chain), we can reduce both sides alternately. As we are not able to find any value that satisfies both sides

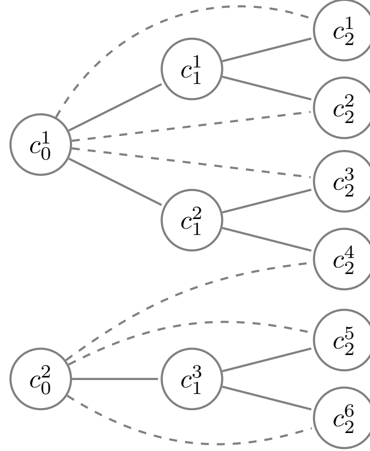


Figure 6.2: Instance satisfying the model $\mathfrak{M}$ from Example 6.2: Black edges are instantiations of associations u and v, dashed edges are instantiations of association w.

of the equation, we will finally end up with either an empty interval for at least one composed association, or with intervals that are no longer overlapping.

Example 6.3. We apply Algorithm 5.6 to the model from Example 6.2 for the equation $E = (uv = w)$. We have $U_{\text{new}} = U = uv$ and $V_{\text{new}} = V = w$ (line 5), hence the composed type of U_{old} is $C_0 \;_{1..1}\!\Rightarrow_{2..4} C_2$ (line 8) and the one of V_{old} is $C_0 \;_{1..1}\!\Rightarrow_{3..3} C_2$ (line 9). Intersecting the association chains we obtain $a..A = 1..1$ (line 10) and $b..B = 2..3$ (line 11). Since the intervals overlap, ChainReduction reduces U_{old} with respect to the intersection type $T : C_0 \;_{1..1}\!\Rightarrow_{2..3} C_2$ (line 16); in particular, the multiplicity $1..2$ of association u is reduced to $1..1$. V_{old} cannot be reduced with respect to T, hence we have $V_{\text{new}} = V_{\text{old}}$ (line 17). Since U_{new} has been modified, the loop is repeated. Recalculating the composition uv in line 8 yields the new type $C_0 \;_{1..1}\!\Rightarrow_{2..2} C_2$. Since the multiplicity $2..2$ does not overlap with $3..3$ of w, the algorithm throws an "unsatisfiable" exception (line 13). Consequently the model is E-unsatisfiable.

Checking E-satisfiability

From the previous chapter we can conclude that if a model $\mathfrak{M}$ is E-satisfiable its reduced version $\mathfrak{M}'$ has to be satisfiable. This is a consequence of the fact that reducing multiplicities preserves E-satisfiability. Therefore a non-failing call to ModelReduction is a necessary requirement for $\mathfrak{M}$ to be E-satisfiable. Our experiments and our futile search for counter-examples suggest that it is also sufficient, leading to the following conjecture.

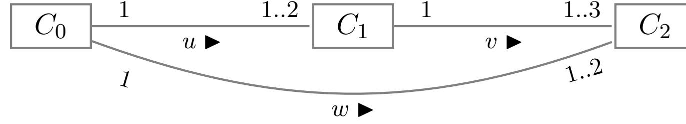
Conjecture 6.4. *Let $\mathfrak{M}'$ be the model obtained from $\mathfrak{M}$ by reducing all multiplicities with respect to its equations. Then $\mathfrak{M}'$ is satisfiable iff $\mathfrak{M}$ is E-satisfiable.*

If this conjecture holds, we can systematically check the E-satisfiability of a model $\mathfrak{M} = (\mathcal{C}, \mathcal{A}, \mathcal{E})$ by first reducing all multiplicities in $\mathcal{A}$ under $\mathcal{E}$ to obtain the reduced model $\mathfrak{M}'$ and

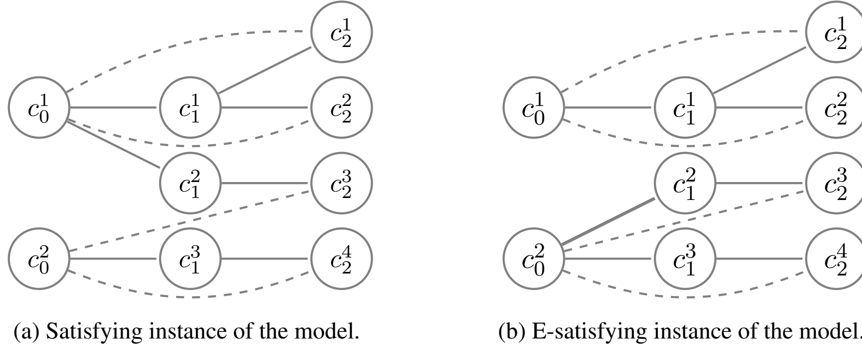
then checking $\mathfrak{M}'$ for satisfiability. This method is sound as the reduced model $\mathfrak{M}'$ has the same E-satisfying instances as $\mathfrak{M}$.

Unfortunately the problem of computing E-satisfying instances can not be reduced to the one of computing satisfying instances by reducing multiplicities. As the following example illustrates, the reduced model $\mathfrak{M}'$ may still have satisfying instances which are no E-satisfying instances of $\mathfrak{M}$ (and $\mathfrak{M}'$).

Example 6.5. Consider the following model $\mathfrak{M} = (\mathcal{C}, \mathcal{A}, \{uv = w\})$:



Reducing this model leads to a model $\mathfrak{M}'$ where v is of type $C_1 \; {}_{1..1}\!\!\Rightarrow_{1..2} \; C_2$. There are instances of the association chain uv which cannot be extended to an E-satisfying instance of $\mathfrak{M}'$ (see Figure 6.3a). Adding further links does not lead to an E-satisfying instance, because the equation $uv = w$ is violated by object c_2^3, which is linked to c_0^1 via uv, but to c_0^2 via w. As the objects c_0^1 and c_0^2 are already linked with the maximal number of partner objects of class C_2 via w, and each object of class C_2 and C_1 must have exactly one partner object of class C_1 (via v) and C_0 (via u), respectively, we are stuck. However, we can find an E-satisfying instance for $\mathfrak{M}'$ by relinking the existing objects accordingly: we replace the link between c_0^1 and c_1^2 with a link between c_0^2 and c_1^2 (see the bold edge in Figure 6.3b).



(a) Satisfying instance of the model.

(b) E-satisfying instance of the model.

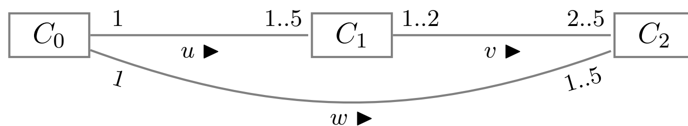
Figure 6.3: Satisfying instances of the model $\mathfrak{M}$ from Example 6.5: Black edges are instantiations of associations u and v, dashed edges are instantiations of association w.

6.2 Minimal satisfying instance under equality constraints

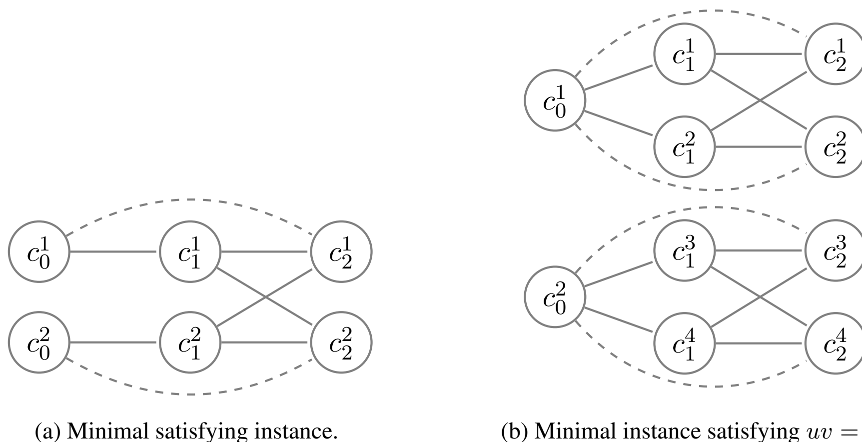
We observed that not every satisfying instance of a reduced model is also an E-satisfying instance. The same holds for the minimal satisfying instance and the minimal E-satisfying instance

of a non-reduced model: the minimal satisfying instance of a model $\mathfrak{M}$ is not necessarily identical to the minimal E-satisfying instance of $\mathfrak{M}$. In general, the minimal E-satisfying instance will be larger regarding the number of required objects and links.

Example 6.6. Consider the following model $\mathfrak{M}$:



Suppose we need at least two objects of each class C_0 and C_2 (e.g. required by a user constraint), and we are given the equality constraint $uv = w$. Then the minimal satisfying instance for $\mathfrak{M}$ (Figure 6.4a) is not E-satisfying as each C_0-object is connected to two different C_2-objects via the composed relation $r_u \circ r_v$ but only to one C_2-object via relation r_w. In contrast, the minimal E-satisfying instance for $\mathfrak{M}$ under $uv = w$ needs four objects of class C_2 (Figure 6.4b).



(a) Minimal satisfying instance. (b) Minimal instance satisfying $uv = w$.

Figure 6.4: Satisfying instances of the model $\mathfrak{M}$ from Example 6.7: Black edges are instantiations of associations u and v, dashed edges are instantiations of association w.

In general, it does not help to reduce the model w.r.t to an equation E to ensure that the minimal satisfying instance is identical to the minimal E-satisfying instance (concerning the number of objects per class).

Example 6.7. The minimal instance E-satisfying the model in Figure 6.5 is larger than the minimal instance that only satisfies the model (see Figure 6.6). The model cannot be reduced any further w.r.t. the equation $u_1u_2u_3 = v$. The intersection of both sides of the equation is of type $C_0\ {}_{1..1}{\Rightarrow}_{2..5}\ C_3$. The upper bound of each association within the association chain $u_1u_2u_3$ can be reached by choosing the lower bounds of both other associations, resulting in a total of four objects of class C_3 connected to a single C_0-object. This number does not exceed the

upper bound of the intersection type and is thus a valid result. The same holds for each lower bound when combined with the upper bounds of the two remaining associations. As the model is already reduced w.r.t. the equation $u_1u_2u_3 = v$, we cannot find a minimal E-satisfying instance of a model by generating the minimal satisfying instance for the reduced model and rearranging the links accordingly.

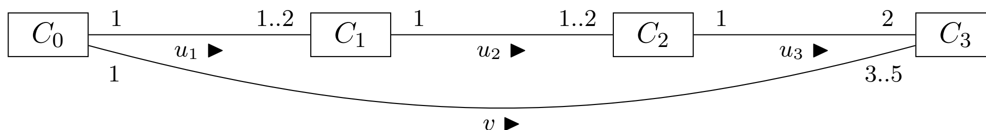
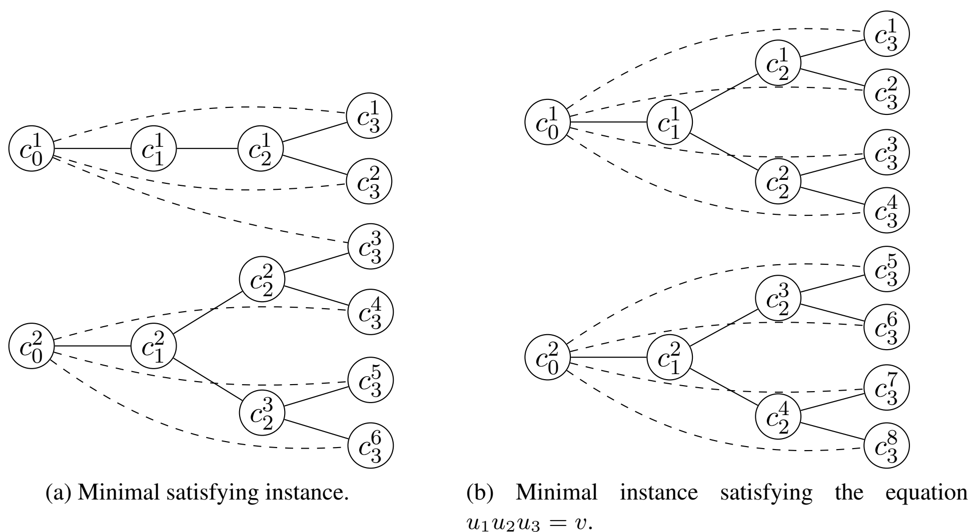


Figure 6.5: Model reduced w.r.t. the equation $u_1u_2u_3 = v$.



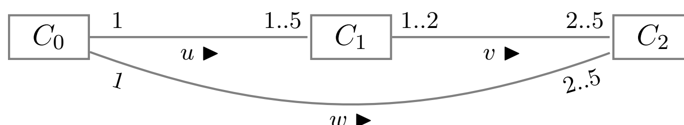
(a) Minimal satisfying instance.

(b) Minimal instance satisfying the equation $u_1u_2u_3 = v$.

Figure 6.6: Satisfying instances of the model in Figure 6.5: Black edges are instantiations of associations u_i, dashed edges are instantiations of association v.

Nevertheless, we conjecture that for reduced models $\mathfrak{M}$ with an equation E that only contains chains of at most two associations, for some minimal satisfying instance $\mathfrak{I}$ of $\mathfrak{M}$ there exists a minimal E-satisfying instance $\mathfrak{I}'$ that is identical to $\mathfrak{I}$. This does not hold for arbitrary minimal instances, as the objects may be linked in a different way.

Example 6.8. Consider the model $\mathfrak{M}'$ obtained by reducing the model from Example 6.7 w.r.t. the equation $uv = w$:



The association w is now of type $C_0 \ {}_{1..1}\!\Rightarrow_{2..5}\ C_2$. The minimal satisfying instance for $\mathfrak{M}$ (Figure 6.4a) is not satisfying $\mathfrak{M}'$, as each C_0-object needs two different C_2-objects via relation r_w, as well as via the composed relation $r_u \circ r_v$. The minimal E-satisfying instance depicted in Figure 6.4b is thus also the minimal satisfying instance of $\mathfrak{M}'$.

We have not yet been able to find any counterexample to this assumption.

6.3 Tree-generating equations

Although we could not yet prove or disprove Conjecture 6.4, we are able to prove a result for a restricted family of models, which is of practical relevance. We call the type of equality constraints that characterise this family of models *tree-generating*.

Definition 6.9 (tree-generating)**.** An association chain $C_0 \ {}_{m_1..M_1}\!\Rightarrow_{n_1..N_1} \cdots {}_{m_k..M_k}\!\Rightarrow_{n_k..N_k}$ C_k is *one-many* if $m_i = M_i = 1$ for all $i = 1, \ldots, k$. An equation $U = V$ is *tree-generating* if the association chains U and V are one-many. A model $\mathfrak{M} = (\mathcal{C}, \mathcal{A}, \mathcal{E})$ is tree-generating, if $\mathcal{C}$ and $\mathcal{A}$ are the sets of classes and associations occurring in $\mathcal{E}$ and all equations in $\mathcal{E}$ are tree-generating.

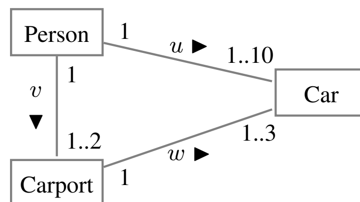


Figure 6.7: Model of persons, their cars and carports.

Example 6.10. Consider the model depicted in Figure 6.7. Both association chains vw and u are one-many, therefore the equation $u = vw$ and hence also the model are tree-generating.

A satisfying instance of a one-many association chain containing exactly one C_0-object is a tree with this C_0-object at its root. Every non-trivial satisfying instance of a one-many association chain consists of one or more such trees. Each C_0-object that is part of a satisfying instance of a one-many association chain $C_0 \ {}_{m_1..M_1}\!\Rightarrow_{n_1..N_1} \cdots {}_{m_k..M_k}\!\Rightarrow_{n_k..N_k}\ C_k$ is the root of a tree. If we consider the tree as a directed graph, C_0 is the starting node. Thus, for each tree-generating equation we obtain two trees starting at the same C_0-object.

Reducing upper bounds for tree-generating equations

For tree-generating equations, we can give an explicit formula for reducing the upper bounds of multiplicities within an association chain. We start with the formula for $g_i(i, \alpha)$ from Proposition 5.9, combined with the calculation for all associations u_j with $j > i$:

$$n_i \cdot (g_{i-1}(i,\alpha) - 1) + \alpha) \cdot \prod_{j=i+1}^{k} n_j \;=\; g_k(i,\alpha) \;\leq\; B \;.$$

Note that this formula differs from the general case, because we have $M_i = 1$ for all i. Now we include all associations u_j with $j < i$ by replacing $g_{j-1}(i,\alpha)$:

$$n_i \cdot (\prod_{j=1}^{i-1} n_j - 1) + \alpha) \cdot \prod_{j=i+1}^{k} n_j \;=\; g_k(i,\alpha) \;\leq\; B \;,$$

which can be rewritten as

$$\prod_{j=1}^{k} n_j + (\alpha - n_i) \cdot \prod_{j=i+1}^{k} n_j \;\leq\; B.$$

and hence

$$\alpha \leq \left\lfloor \frac{B + \prod_{j=i}^{k} n_j - \prod_{j=1}^{k} n_j}{\prod_{j=i+1}^{k} n_j} \right\rfloor \;.$$

The reduced upper bound N_i' of association u_i, can thus be calculated using the explicit formula

$$N_i' = \min\left(B \cdot \prod_{j=i+1}^{k} \frac{1}{n_j} + n_i - \prod_{j=1}^{i} n_j \;,\; N_i \right) \;. \tag{6.1}$$

E-satisfiability under tree-generating equations

For tree-generating models (i.e. consisting only of association chains that form one or more tree-generating equations), we can check E-satisfiability with the help of the inequalities we get from the model (see Section 2.3).

Proposition 6.11. *Let $\mathfrak{M} = (\mathcal{C}, \mathcal{A}, \mathcal{E})$ be a model such that $E = (U{=}V)$ in $\mathcal{E}$ is a tree-generating equation and $\mathcal{C}$ and $\mathcal{A}$ are the sets of classes and associations occurring in E. Let C be the initial class of the association chains U and V. Then $\mathfrak{M}$ is E-satisfiable if and only if both, U and V, have a satisfying instance with $|C| = 1$ and all classes occurring in both chains are of the same cardinality.*

The existence of such instances can be checked by solving the corresponding inequalities for $|C| = 1$.

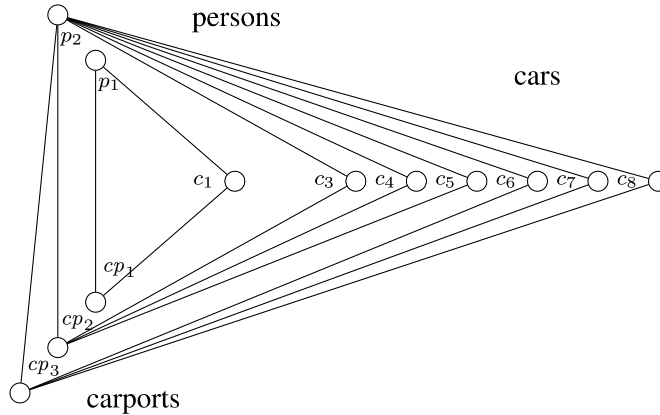


Figure 6.8: Instance where every car is parked in the carport of the person it belongs to.

Proof. Due to the multiplicities $1..1$ at one association end all E-satisfying instances of $\mathfrak{M}$ are forests with single C-objects as root. Therefore both chains have a satisfying instance with the same cardinality of shared classes. On the other hand, given such instances for a single C-object we identify the objects of shared classes and obtain an E-satisfying instance for one C-object. By taking the union of $|C|$ independent copies we obtain an E-satisfying instance of $\mathfrak{M}$. $\square$

We now want to check whether our Carport-example (Example 5.1) is in fact E-satisfiable.

Example 6.12. For the model $\mathfrak{M} = (\mathcal{C}, \mathcal{A}, \mathcal{E})$ in Figure 6.7, we compute the reduced model $\mathfrak{M}'$ with respect to $u = vw$ (Figure 5.6), which tightens the interval $1..10$ to $1..6$ in the association u (Example 5.11). If MODELREDUCTION had failed, we would have concluded that the model is not E-satisfiable. Since the equation $u = vw \in \mathcal{E}$ is tree-generating (Example 6.10), we apply Proposition 6.11. Obviously both chains, u and vw, admit instances with one person and one to six cars. Therefore $\mathfrak{M}$ is E-satisfiable. We obtain a satisfiable instance for $\mathfrak{M}'$ by taking the instance from Figure 5.2 and deleting the object c_2 and all corresponding links. The resulting instance is shown in Figure 6.8).

Minimal Instances under tree-generating equality constraints

The minimal E-satisfying instance and the minimal satisfying instance are not even identical for reduced tree-generating models. Reconsider the model in Figure 6.5, which is tree-generating. In Example 6.7 we observed that the minimal E-satisfying instance of the model is larger than the minimal instance that only satisfies the model (see Figure 6.6).

7 Linking Objects with Netflow Algorithms

If the facts don't fit the theory,
change the facts.

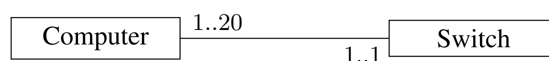
Albert Einstein

This chapter shows how to distribute links and how to repair configurations (i.e. instances) with the help of flow networks. As this is a relevant topic in the field of configuration management, we will mainly use the term "configuration" as a synonym for "instance" (see Chapter 3).

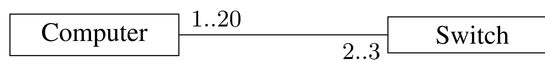
Note that UML distinguishes between *unique* and *non-unique* associations (or mixed associations where one association end is *unique* and the other one is *non-unique*). The multiplicities of an association end labelled as *unique* restrict the number of partner objects, whereas those of an association end labelled as *non-unique* restrict the number of links to (not necessarily distinct) partner objects. In this chapter we will consider both types of uniqueness constraints (but no mixed associations). This approach does not include equality constraints. Hence a model is defined as $\mathfrak{M} = (\mathcal{C}, \mathcal{A})$ (see Chapter 2). We consider only single associations. The approach can be extended to models consisting of more than one association by composing the results (i.e. the link distributions), as the distribution of links is compositional.

7.1 Motivation

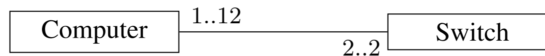
Suppose a company has a network of computers connected by switches. Each switch can be connected to between one and 20 computers and each computer has to be connected to one switch.



Now the company grows and we need additional computers. At some point, we have to buy additional switches. Furthermore, we want to build a failure-safe network and therefore force each computer to be connected to two or three switches, instead of only one. Of course, we want to do as little rewiring as possible, but at the same time keep the number of switches as small as possible to reduce the costs.



Later, the company wants to replace all switches by a new model that has only 12 ports. The replaced switches would now need to serve up to 20 computers, which is too much for 12 ports. Furthermore, the company wants to reduce the number of switches a computer is connected to, thus forcing each computer to be connected to exactly 2 switches. In this scenario, we have to remove some wires and/or rewire them, again trying to keep as many existing wires as possible.



For such changing requirements, a tool that gives us the minimally required switches for our network and also computes a conservative rewiring would be very helpful, i.e., reconfiguration and configuration repair is of central importance.

As mentioned in Chapter 3, one of the challenges in configuration management is the task of completing or repairing configurations when requirements change. This happens if either the specification changes, or objects and/or links are added to or deleted from an existing configuration. In these cases we need to find a conservative, but still cost-effective extension of the configuration.

When checking the satisfiability of a UML class diagram with the help of an ILP-solver, we get the number of objects of each class needed for a minimal instance of the model and a lower and an upper bound for the number of links to instantiate the associations. How to distribute those links still needs to be discussed. Our approach to finding a link distribution such that the corresponding configuration satisfies a given specification is to solve a minimum cost flow on an adequate network. As the distribution of links is compositional, we can deal with each association separately.

The advantages of flow networks are:

- We may use the same ILP solvers we used for the formal verification and optimisation of models.
- Minimum cost flow problems are solvable in polynomial time.
- The method is applicable to configuration completion and repair.
- The approach is highly flexible.

We start by introducing flow networks and the minimum cost flow problem, which we will use in the following sections to distribute links in various applications.

7.2 Flow Networks and the Minimum Cost Flow Problem

Flow networks are used to model problems like the *transportation problem*, where goods are sent from origins to destinations via routes that have certain capacities and costs. Origins and

destinations are modelled as nodes and routes as directed edges between the nodes. Goods are represented by units that flow along paths from origins to destinations. Flow networks model physical networks like oil pipelines or communication or electrical networks, but can as well be used for more abstract applications like scheduling problems. Typical problems are to maximise the flow through a network (*Maximum Flow Problem*) or to minimise the cost (*Minimum Cost Flow*). A detailed overview can be found in [1].

Formally, a *flow network* is a directed graph (V, E), where V is a set of nodes (or vertices) and $E \subseteq V \times V \times N$ is a set of directed edges (or arcs) [1]. Each edge (i, j, p) is identified by its starting node i, its destination node j and a value p, denoting that it is edge number p between nodes i and j. If there is only one edge between nodes i and j, we will write (i, j) instead of $(i, j, 1)$. Every node i has a value $b(i)$ assigned. A *source* is a node i with a supply $b(i) > 0$, whereas a *target* (also called *destination* or *sink*) is a node i with a demand $b(i) < 0$. *Transit nodes* are nodes with neither supply nor demand, i.e., we have $b(i) = 0$ for all transit nodes i. Furthermore, in a flow network we have lower and upper bounds, l_{ij} and u_{ij}, on the flow over each edge (i, j), and a cost c_{ij} for transporting one unit of flow along the edge (i, j). In the case of multiple edges we write l^p_{ij}, u^p_{ij} and c^p_{ij}. A *flow* is a function $f\colon E \mapsto \mathbb{N}$; we will use f_{ij} instead of $f((i, j))$ for single edges and likewise f^p_{ij} instead of $f((i, j, p))$ for multiple edges. A flow is *feasible* if it satisfies the following constraints

$$\sum_{\{(j,p):(i,j,p)\in E\}} f^p_{ij} - \sum_{\{(j,p):(j,i,p)\in E\}} f^p_{ji} = b(i) \qquad \text{for all } i \in V \tag{7.1}$$

$$l^p_{ij} \leq f^p_{ij} \leq u^p_{ij} \qquad \text{for all } (i, j, p) \in E \ . \tag{7.2}$$

$$f^p_{ij} \text{ is integer for all } (i, j, p) \in E \tag{7.3}$$

The constraints in (7.1) specify that the flow leaving a node (the *outflow*) minus the one entering it (the *inflow*) has to equal the supply or demand at that node; they are called *mass balance constraints*. The constraints in (7.2), called the *flow bound constraints*, require that the flow along each edge is within the given bounds (lower bound l^p_{ij} and capacity u^p_{ij}). The *integrality constraints* in (7.3) force all flows to be integer valued. Furthermore we assume that all data (capacities, costs and supplies/demands) are integral.

Subject to the constraints (7.1)- (7.3), the *Minimum Cost Flow problem* minimises the total costs, $\sum_{(i,j,p)\in E} c^p_{ij} f^p_{ij}$.

MINCOSTFLOW
Input: A flow network with values $b(i)$, l^p_{ij}, u^p_{ij}, and c^p_{ij} for all $i \in V$ and all $(i, j, p) \in E$.
Output: A feasible flow f such that $\sum_{(i,j,p)\in E} c^p_{ij} f^p_{ij}$ is minimal.

Minimal cost flows can be computed in polynomial time [1]. One approach that fits our framework particularly well is to map the problem to integer linear programming. This way we can use the same tools for solving the inequalities of section 2.3 and for computing minimal flows.

Throughout this thesis, we specify flow networks with all necessary properties in graphical form. Each arc of the network is labelled as depicted in Figure 7.1, where

- l_{ij} is the *lower capacity* of edge (i, j) from node i to node j
- u_{ij} is the *upper capacity* of edge (i, j)
- c_{ij} is the *cost* of sending one unit of flow from node i to node j.
  This component is omitted if it is the same for all $(i, j) \in E$.

If the capacity bounds of an edge are 0 and ∞, the edge may be labelled as (c_{ij}).

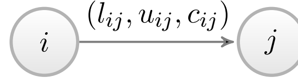


Figure 7.1: Labelling of arrow from node i to node j

A node i is labelled with i if it is a transit node (i.e. it has neither a supply nor a demand of flow), and by $i[+b]$ or $i[-b]$ if it has a supply or demand of b units of flow.

For multiple edges between a pair of objects (i, j) we label each edge (i, j, p) with l_{ij}^p, u_{ij}^p and c_{ij}^p (according to the definition).

7.3 Distributing Links

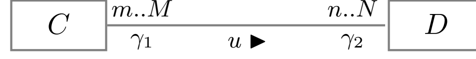
LINKDISTRIBUTION is the task of finding a set of links $\mathcal{L}$ such that $\mathfrak{I} = (\mathcal{O}, \mathcal{L})$ is a (minimal) instance satisfying a given model $\mathfrak{M}$. The set of objects $\mathcal{O}$ instantiating $\mathfrak{M}$ is part of the input.

LINKDISTRIBUTION
Input: a model $\mathfrak{M}$ and a set of objects $\mathcal{O}$
Output: a set of links $\mathcal{L}$, such that the instance (configuration) $\mathfrak{I} = (\mathcal{O}, \mathcal{L})$ satisfies $\mathfrak{M}$ (and is optimal in a sense to be specified)

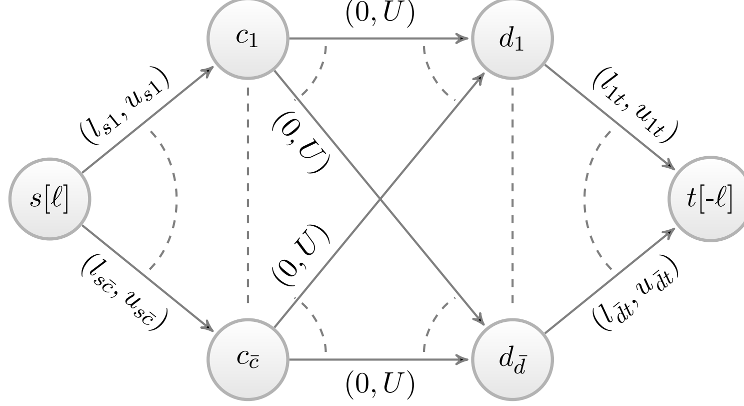
A minimal satisfying instance for a model can be constructed as follows:

1. Translate the model to inequalities forming an ILP problem (see Section 2.3).
2. Solve this ILP problem to determine the minimal number of objects necessary for a satisfying instance. Fix the number of links between the given bounds (e.g. choose the smallest number).
3. Solve the corresponding netflow problem for each association to arrange the links in an admissible way.

Figure 7.2b shows a flow network corresponding to an instance of the association in Figure 7.2a with $\bar{c}$ C-objects, $\bar{d}$ D-objects and ℓ links. The source node s is connected to every node c_i and each node d_j is connected to the target node t. Furthermore there exists an edge from each node c_i to every d_j. The cost c_{ij} is left unspecified (or can be set to zero) for all edges (i, j). The general notation conventions are described in section 7.2. The lower and upper



(a) Binary association.



(b) 4-layered flow network to distribute links for association u.

Figure 7.2: Association and its corresponding flow network for $\bar{c}$ D-objects and $\bar{d}$ D-objects.

bounds for the capacities of edges from node s to nodes c_i are given as l_{si} and u_{si} and those of edges from nodes d_j to node t as l_{jt} and u_{jt}.

To ensure that the links are distributed correctly (i.e. respecting the constraints imposed by the underlying model), we have to choose the number of links and the capacities of the edges accordingly:

- Use the inequalities from Section 2.3 to find bounds for the number of links: $\ell_{\min} \leq \ell \leq \ell_{\max}$
- Select one particular ℓ, e.g. the minimal one, and use it as supply at node s and as demand at node t
- Capacity bounds from c_i to d_j: for all i, j fix a lower bound of zero for the flow from c_i to d_j and an upper bound of $U = 1$ for $\gamma_1 = \gamma_2 = \textit{unique}$ and of $U = \ell$ for $\gamma_1 = \gamma_2 = \textit{nonunique}$.

Note that the upper bounds on edges between the C- and D-objects (U) are different for *unique-unique* and *nonunique-nonunique* associations (i.e. $\gamma_1 = \gamma_2 = \textit{unique}$ resp. $\gamma_1 = \gamma_2 = \textit{nonunique}$). For the *unique* case, we choose $U = 1$, as there can be only one link between each pair of objects. If we have a *nonunique-nonunique* association, there can be more than one link between each pair of objects, namely at most $min(N, M)$. We choose $U = \ell$, as we will restrict the flow to the allowed value by setting u_{si} and u_{jt} accordingly. The lower and upper capacity bounds of edges leaving s and of those entering t depend on whether we want a uniform link distribution or not. We will investigate both cases in the following subsections.

A flow of f_{ij} on arc (i, j) means that we have to place f_{ij} links between objects c_i and d_j in the corresponding configuration.

Definition 7.1 (Link distribution corresponding to a flow)**.** The link distribution corresponding to a flow f is the distribution containing f_{ij} links between objects c_i and d_j.

The link distribution for a model $\mathfrak{M} = (\mathcal{C}, \mathcal{A})$ is obtained by calculating the link distribution of each association in $\mathcal{A}$.

Uniform Link Distribution

If we want to build a minimal instance without any further restrictions on the objects and links, we can distribute the links uniformly among the objects. This particular network was proposed in [34].

To ensure that the links are uniformly distributed, we have to choose the capacities of the edges in the flow network accordingly (see Figure 7.3):

- Choose $l_{si} = \lfloor \frac{\ell}{\bar{c}} \rfloor$ and $u_{si} = \lceil \frac{\ell}{\bar{c}} \rceil$
- Choose $l_{jt} = \lfloor \frac{\ell}{\bar{d}} \rfloor$ and $u_{jt} = \lceil \frac{\ell}{\bar{d}} \rceil$

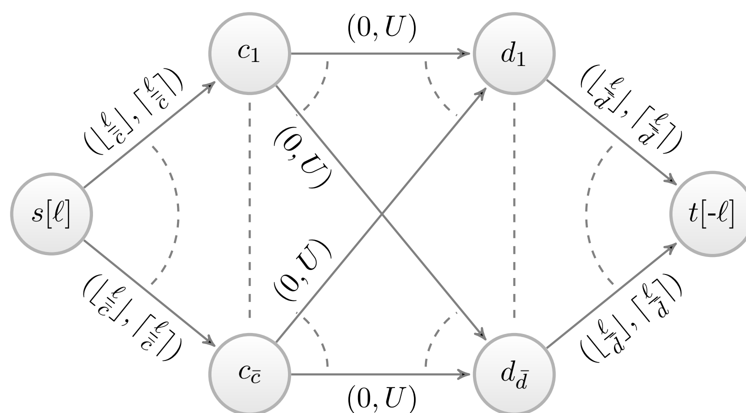


Figure 7.3: Flow network for uniform link distribution.

By dividing the links by the number of C-objects ($\bar{c}$), we distribute the links uniformly between all C-objects. As we are only dealing with integers, we might need to round the result, thus giving us the rounded result as bounds for the number of links for each object (the rounded down result as a lower bound and the rounded up result as an upper bound). We do the same for the D-objects.

The link distribution corresponding to a feasible flow in this network is a uniform distribution satisfying the association. By computing a feasible flow for each association of a model and taking the composition of all corresponding link distributions, we get a configuration satisfying the model.

At this point the use of flow networks is not yet an advance, since it is always possible to construct a satisfying configuration by distributing the links uniformly among the objects using *balanced sequences* (see [34] and [21] for a detailed description of this approach). However, by using different capacity bounds and non-zero costs we are able to model various scenarios in the next sections.

General link distribution

As described above, we can use netflow algorithms to distribute links uniformly among a given set of objects. So far, links had no costs assigned (or all links had the same cost). However, in reality this might not be the case. A uniform distribution of links in a scenario where links between some pairs of objects are cheaper than others might lead to a solution that is not optimal concerning the costs. Hence, we would like to adapt the approach to be able to deal with different types of links.

In fact, a uniform distribution of links is more restrictive than necessary. It is sufficient to ensure that the multiplicities of the association we deal with are not violated. So we need to find an instantiation of our network from Figure 7.2b that ensures the multiplicities are respected. We have to adapt the capacities accordingly (see Figure 7.3):

- Choose $l_{si} = n$ and $u_{si} = N$
- Choose $l_{jt} = m$ and $u_{jt} = M$,

where (n, N) are the multiplicities restricting the number of (links to) partner objects of each C-object and (m, M) are the multiplicities restricting the number of (links to) partner objects of each D-object (see Figure 7.2a).

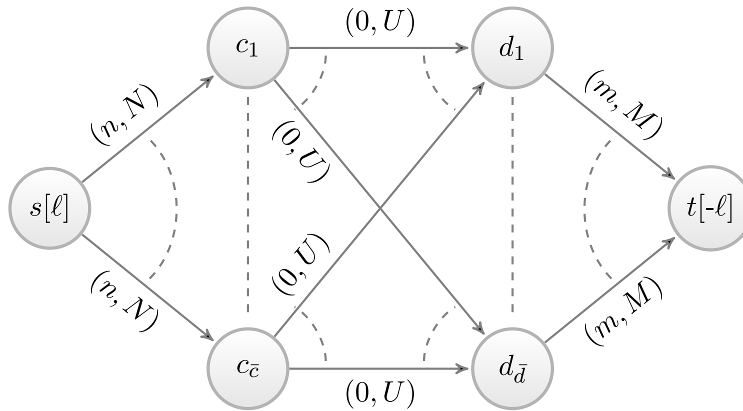


Figure 7.4: Flow network for general link distribution.

Using this network, we can state that the link distribution corresponding to a feasible flow in the flow network of each association of a model together form a satisfying instance of the model and that the link distributions for each association of a satisfying instance correspond to a feasible flow in the respective flow network.

Proposition 7.2. *For a given model let N_a be the network corresponding to each association a (as described above), with arbitrary costs. Then an instance satisfies the model if and only if all corresponding flows in the networks N_a are feasible.*

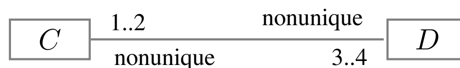
Proof. In a network N_a corresponding to an association a the flow entering each node of class C has to respect the flow bounds n, N. This guarantees that, for a feasible flow in the network, the number of D-objects connected to each C-object lies between n and N, which corresponds to the restrictions imposed by the multiplicities. The flow leaving each node of class D has to respect the flow bounds m, M. For a feasible flow, each D-object is therefore connected to at least m and as most M C-objects. This interval corresponds to the multiplicities restricting the number of partner objects for each D-object.

The link distribution corresponding to a feasible flow in the network N_a hence results in an instantiation of association a that satisfies the specification.

The link distribution of each association is independent from the link distributions of all other associations of the model. Combining the link distributions corresponding to a feasible flow in each network N_a therefore leads to an instance that satisfies the model. $\square$

Thus, we can solve the LINKDISTRIBUTION problem by first computing a valid number of objects for each class and a number of links for each association with ILP and afterwards solving a minimum cost flow problem for each association on the corresponding network. The link distributions corresponding to a feasible flow in the network of each association together form the link distribution of the configuration.

Example 7.3. Consider the following *nonunique*-*nonunique* association:



By solving the corresponding inequalities with ILP we obtain the minimally required number of objects for both classes and a range for the number of links: one object of class C and two objects of class D and $[\ell_{\min}, \ell_{\max}] = [3, 4]$. As we are interested in building the minimal instance, we choose the number of links to distribute as $\ell = \ell_{\min} = 3$. The corresponding flow network is depicted in Figure 7.5. Solving a MINCOSTFLOW problem on this network results in the flow $f_{11} = f_{12} = f_{13} = 1$. The link distribution corresponding to this flow contains one link from object c_1 to each object of class D. The resulting minimal instance is depicted in Figure 7.6.

In the following sections, we discuss several variations of the LINKDISTRIBUTION problem.

7.4 Completing Configurations

To complete a configuration (i.e. an instance) means to extend an existing configuration (without modifying the existing objects and links), such that it becomes a satisfying instance of a given model.

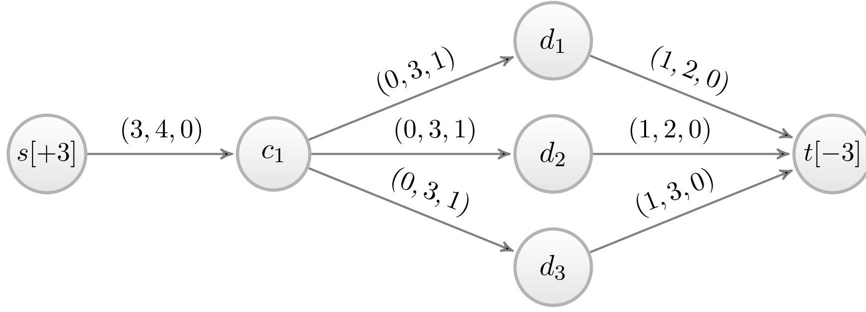


Figure 7.5: Flow network for finding a link distribution. The minimal cost is obtained for the flow $f_{11} = f_{12} = f_{13} = 1$.

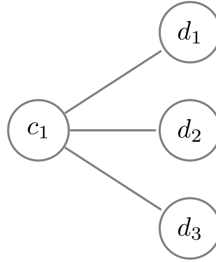


Figure 7.6: Minimal instance of the association in Example 7.3.

CONFIGURATIONCOMPLETION
Input: a model $\mathfrak{M}$ and a configuration $\mathfrak{I} = (\mathcal{O}, \mathcal{L})$ that is an instance of $\mathfrak{M}$
Output: a (minimal) configuration $\mathfrak{I}' = (\mathcal{O} \cup \mathcal{O}', \mathcal{L} \cup \mathcal{L}')$ that satisfies $\mathfrak{M}$

Such problems arise when an existing configuration shall be extended, for example by adding additional computers to an existing network. They also arise when the model changes, e.g. by requiring that more computers have to participate in the network. Of course, this results in an extension of the existing configuration as well, as the minimal instance changes. The aim here is to maintain the current configuration and to just extend it such that it satisfies the model. It is usually not desirable to view the problem as the search for a minimal instance of the new model, as this might lead to a complete rearrangement of the old components and links.

In our framework of ILP and flow networks this problem can be solved as follows.

1. Solve the inequalities corresponding to the model $\mathfrak{M}$ to determine the number of objects and links (interval $[\ell_{\min}, \ell_{\max}]$) required by a minimal instance or by an instance containing at least the objects in $\mathcal{O}$.

2. If more objects are needed than available in $\mathcal{O}$, add an appropriate number, giving $\mathcal{O}_{new} = \mathcal{O} \cup \mathcal{O}'$.

3. For the number of links, ℓ, take the minimum of $\ell_{\min}$ (computed in the first step) and $|\mathcal{L}|$.

4. Construct a flow network as described in Figure 7.2b. If $\mathcal{L}$ contains l links between objects c_i and d_j, set the lower bound of the corresponding edge in the network to l. In the case of the attribute unique this means that the lower as well as the upper bound on this edge will be 1. Additionally, reduce the flow bounds on the edges to nodes c_i and from nodes d_j by l.

   If this results in a negative lower bound on some edge, there is no extension containing the original network, no matter how many other objects or links we add. In this case we need to repair the configuration instead (see Section 7.5).

5. If a feasible flow exists, solve a MINCOSTFLOW problem on the network and the problem is solved: We have found a minimal extension of the original configuration satisfying the new model.

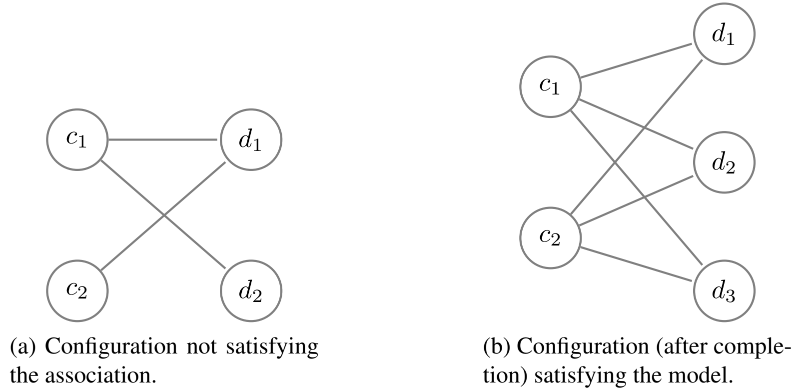
   If no feasible flow exists, increase ℓ and repeat this step. If no costs are used on the edges, then binary search can be used to find the minimal ℓ leading to a solvable flow problem.

   If there is no feasible flow for any value ℓ up to the upper bound computed by ILP, then there is no extension with the computed number of objects that contains the original configuration.

Example 7.4. Consider the following *nonunique*-*nonunique* association:

Let the configuration $\mathfrak{I} = (\mathcal{O}, \mathcal{L})$ be an instance of this association with $\mathcal{O} = \{c_1, c_2, d_1, d_2\}$ and $\mathcal{L} = \{(c_1, d_1), (c_1, d_2), (c_2, d_1))\}$ (see Figure 7.7a). By solving the corresponding inequalities with ILP we obtain a new set of objects $\mathcal{O} \cup \mathcal{O}' = \{c_1, c_2, d_1, d_2, d_3\}$ and a range for the number of links: $[\ell_{\min}, \ell_{\max}] = [6, 6]$. In this case there is only one possible value for the number of links: $\ell = 6$. The corresponding flow network is depicted in Figure 7.8. Edges between nodes that represent linked objects (e.g. edge (c_1, d_1)) have a lower capacity bound of 1, thus forcing the flow algorithm to maintain these links. All edges between nodes c_i and d_j have cost 1, but could as well have any other (non-negative) cost assigned. As long as all edges that represent possible links have the same cost, every pair of objects will be selected with the same probability. Solving a MINCOSTFLOW problem on this network results in the flow $f_{1,1} = f_{1,2} = f_{1,3} = f_{2,1} = f_{2,2} = f_{3,3} = 1$. The link distribution corresponding to this flow contains one link from each C-object to each D-object, i.e. $\mathcal{L} \cup \mathcal{L}' = \{(c_1, d_1), (c_1, d_2), (c_1, d_3), (c_2, d_1), (c_2, d_2), (c_2, d_3)\}$. The resulting configuration (see Figure 7.7b) is a satisfying instance of the association.

If the above procedure fails to find an extension, we can increase the number of objects beyond the minimum computed by the ILP. In the computer network example (see Section 7.1 this could mean that we have to buy another switch in order to avoid recabling the existing network. However, if the cost of new components exceeds the cost of changing links, it is preferable to weaken the constraint that the original configuration has to be maintained by all means. This leads to the problem of configuration repair that can also be addressed in our framework.



(a) Configuration not satisfying the association.

(b) Configuration (after completion) satisfying the model.

Figure 7.7: Instances of the association in Example 7.4.

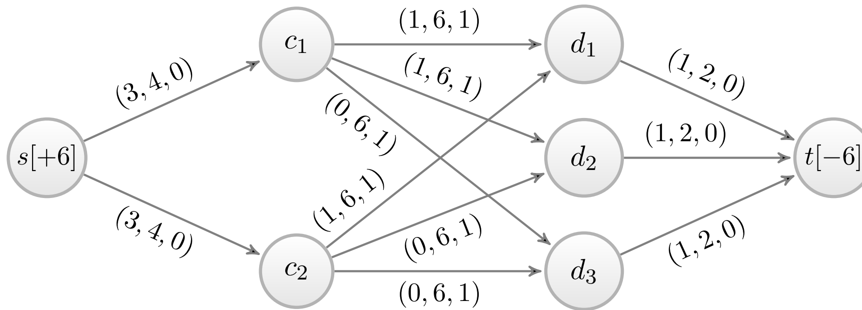


Figure 7.8: Flow network for completing the configuration in Figure 7.7a.

7.5 Repairing Configurations

To repair a configuration means to change a configuration with as few changes as possible such that it becomes a satisfying instance of a given model. The approach described in this section is one possibility to tackle the MINIMALREPAIR problem stated in Section 3.2.

CONFIGURATIONREPAIR
Input: a model $\mathfrak{M}$ and a configuration $\mathfrak{I} = (\mathcal{O}, \mathcal{L})$ that is an instance of $\mathfrak{M}$
Output: a configuration $\mathfrak{I}' = (\mathcal{O} \cup \mathcal{O}', \mathcal{L}')$ satisfying $\mathfrak{M}$ such that $\mathcal{L} \cap \mathcal{L}'$ is maximal.

In the worst case the procedure for extending configurations as described in the previous section leads to an unsatisfiable problem, as existing links must not be removed from the configuration.

A more moderate way to preserve existing links is to keep the capacity bounds unchanged and adapt the costs instead. By using negative costs for existing links, any algorithm for solving the minimal cost flow problem will try to use those arcs, as this reduces the overall costs. This approach has the advantage that an existing link can be removed if keeping it makes the problem unfeasible.

To solve the ConfigurationRepair problem we start with the first three steps of the procedure in the last section, but continue with a different network.

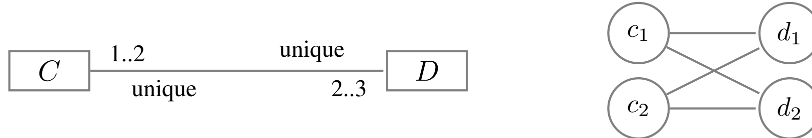
1.–3. See Section 7.4 on configuration completion.

4. Construct a flow network as described in Figure 7.2b. In the unique case, assign cost c_{ex} (e.g. -1) to edges that correspond to links in $\mathcal{L}$, and cost c_{std} (e.g. 1) to all other edges. In the non-unique case the situation is more complex, as some but not all of the possible links between two objects may exist, i.e., in general we have two so-called link priorities. Therefore, if $\mathcal{L}$ contains l links between objects c_i and d_j, introduce two edges between the nodes c_i and d_j. The first edge between c_i and d_j is labelled with $(0, l, c_{ex})$, and the second one with $(0, \max(U - l, 0), c_{std})$.

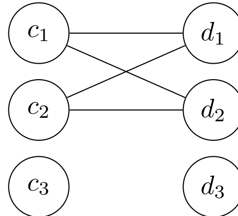
5. A feasible flow always exists (Proposition 7.2). Hence the MinCostFlow problem can be solved, resulting in a configuration that resembles the original one but satisfies the model.

Strictly speaking, this procedure does not solve ConfigurationRepair literally, as there is no guarantee that the number of links shared with the original is maximal. We can approach the solution, however, by increasing ℓ within the bounds computed by Ilp, and we can increase the penalty of introducing new links by decreasing costs for existing links and increasing costs for new ones. In Section 7.8 we will have a closer look at the effect of choosing different costs when repairing configurations.

Example 7.5. Consider the following *unique-unique* association and the configuration $\mathfrak{I} = (\mathcal{O}, \mathcal{L})$ with $\mathcal{O} = \{c_1, c_2, d_1, d_2\}$ and $\mathcal{L} = \{(c_1, d_1), (c_1, d_2), (c_2, d_1), (c_2, d_2)\}$, which is a satisfying instance of the association:

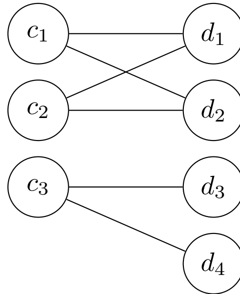


If we want to add a third C-object c_3 to $\mathfrak{I}$, we have to add another D-object as well. This follows from Inequality (4.5), which states that the number of D-objects connected to $x = 3$ C-objects has to be at least $\max(\lceil \frac{nx}{M} \rceil, n \cdot \mathrm{sgn}(x)) = \max(\lceil \frac{2 \cdot 3}{2} \rceil, 2 \cdot \mathrm{sgn}(3)) = \max(3, 2) = 3$.

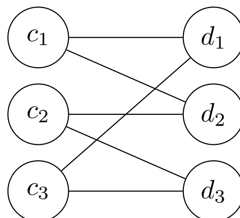


If we are not allowed to remove any existing links at all (as was the case for ConfigurationCompletion in the previous section), we can only connect c_3 to d_3, as d_1 and d_2 are

already connected to the maximally allowed number of C-objects. Therefore, we would have to add a fourth D-object, d_4, and connect c_3 to d_3 and d_4,resulting in the following configuration:



If redirecting existing links is preferable to adding additional objects, we use the CONFIGURATIONREPAIR approach described in this chapter. We already determined the new set of objects: $\mathcal{O} \cup \mathcal{O}' = \{c_1, c_2, c_3, d_1, d_2, d_3\}$. The range for the number of links is $[\ell_{\min}, \ell_{\max}] = [6, 6]$, hence the only possible value for the number of links is $\ell = 6$. The corresponding flow network is depicted in Figure 7.9. Edges between nodes that represent linked objects (e.g. edge (c_1, d_1)) are assigned a negative cost (in this case -1), all other edges between nodes c_i and d_j have cost 1, but could as well have any other (non-negative) cost assigned. All edges between C- and D-nodes have a lower capacity bound of 0, thus not forcing the flow algorithm to choose any specific edge. One possible result of solving a MINCOSTFLOW problem on this network is the flow $f_{11} = f_{12} = f_{21} = f_{23} = f_{31} = f_{33} = 1$. The link distribution corresponding to this flow contains the following links: $\mathcal{L}' = \{(c_1, d_1), (c_1, d_2), (c_2, d_2), (c_2, d_3), (c_3, d_1), (c_3, d_3))\}$. The link (c_2, d_1) is removed from the original configuration and the links (c_2, d_3), (c_3, d_1) and (c_3, d_3) are added. The resulting configuration is a satisfying instance of the association:



Note that we can influence the order in which existing links are removed by setting the costs accordingly.

To solve the CONFIGURATIONREPAIR problem for *nonunique-nonunique* associations we introduce two edges between some pairs of nodes. This procedure is a special case of *priority links*. The following section deals with priority links in general.

7.6 Priority Links

If the cost for links between the same pair of objects varies with the number of links, we need to attach different priorities to them. We call links with priorities attached to them *priority links*.

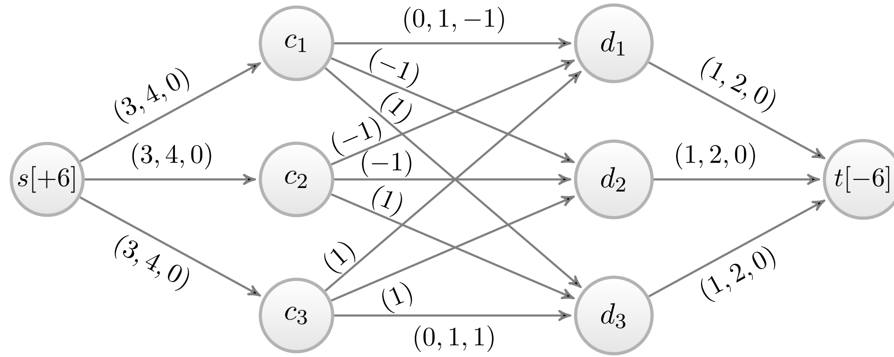


Figure 7.9: Flow network for repairing the configuration given in Example 7.5. Edge labels (1) and (-1) are abbreviations for $(0, 1, 1)$ and $(0, 1, -1)$.

Suppose the first h_1 links between objects c_i and d_j have cost c_{ij}^1, the next h_2 links have cost c_{ij}^2, and so on.

This scenario can be transferred to flow networks by varying the flow costs on an arc between two nodes i and j with the amount of flow. In the network, the first h_1 units of flow between nodes c_i and d_j can be sent for cost c_{ij}^1, the next h_2 units for cost c_{ij}^2, and so on. Such situations can be modelled by *convex cost functions*.

Convex Cost Functions

In cases, where the flow cost on an arc between two nodes i and j varies with the amount of flow, we have to use convex cost functions.

Although it is also possible to use *concise functions*, where the costs of arcs are given in a functional form, we will only deal with *piecewise linear costs*. The reason is that we use flow algorithms to calculate links between objects, hence our data are integral.

In piecewise linear models, the cost on each arc, $C_{ij}(f_{ij})$, consists of k linear segments. A piecewise linear function is specified by its breakpoints $0 = d_{ij}^0 < d_{ij}^1 < d_{ij}^2 < \ldots$ and the slopes of the linear segments between successive breakpoints [1]. c_{ij}^p is the linear cost coefficient of the interval from d_{ij}^{p-1} to d_{ij}^p. Without loss of generality we assume that $c_{ij}^1 < c_{ij}^2 < \cdots < c_{ij}^k$.

The corresponding optimisation problem (for single arcs between all node pairs $(i, j) \in A$) was formulated by Ahuja et al in [1]:

$$\text{Minimise} \qquad \sum_{(i,j)\in E} C_{ij}(f_{ij}) \tag{7.4}$$

subject to the constraints (7.1), (7.2), and (7.3) (see Section 7.2).

Piecewise linear costs can be understood as multiple arcs between a pair of objects. For k linear segments for the cost of edge (i, j) we use k arcs from node i to node j, each arc corresponding to a segment. In Figure 7.10 the first h_1 units of flow can be sent for cost c_{ij}^1, the next h_2 units for cost c_{ij}^2, and so on. Finally, the last h_k units can be sent for cost c_{ij}^k.

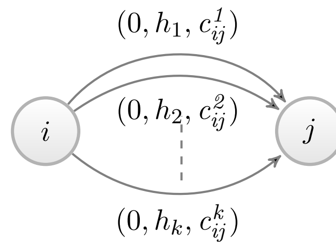


Figure 7.10: Illustration of k cost segments between nodes i and j.

Types of Priority Links

Three different types of constraints might occur in our flow networks.

- *General priority links*: There exist different priorities for links in general, not depending on specific C- or D-objects. This scenario occurs e.g. if a few cables are in store (they can be used for free) and the rest has to be purchased (costing money). It is irrelevant which objects get connected by the free cables.
- *Object-related priority links*: There exist different priorities for links connected to a specific C-object. As an example consider a switch with cables connected to it (but not connected to any specific computer). The links are related to a specific C-object, but not to a specific D-object.
- *Object-pair-related priority links*: In this case there are different priorities per pair of objects (c_i, d_j). This occurs if, for example, some links already exist between two objects c_i and d_j, which should be maintained, if possible. Hence, they are assigned low cost.

Such constraints can either be modelled by convex cost functions or by introducing additional network layers and additional arcs. We will use the second approach to demonstrate how the different types of priority links can be integrated into a flow network.

General Priority Links

To model general priority links, an additional network layer is introduced. This layer consists of one node C that represents all existing C-objects and is placed between the source s and the c_i-nodes. For each priority class we add one arc from s to C with the corresponding cost and capacity. All c_i-nodes are connected to C with capacity bounds n and N and cost 0. Figure 7.11 depicts the corresponding network.

This type of links only reduces the overall cost of each solution by a constant value. The order of the solutions (regarding the costs) and hence the solution of the minimal cost flow are not influenced.

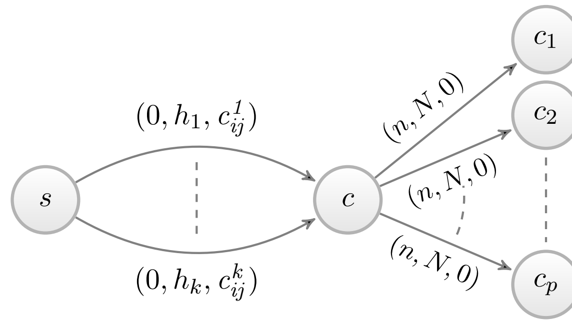


Figure 7.11: Several link priorities for all C-objects

Object-related Priority Links

Object-related priority links from a node c_i are modelled by an additional node c_i' in front of c_i. There are multiple edges between c_i' and c_i (one for each priority class). By connecting s with c_i' by an edge with capacity bounds of n and N we ensure that the multiplicities are still satisfied for object c_i. The resulting network is shown in Figure 7.12. The first h_1 units of flow via c_i have cost c_{ij}^1, the next h_2 units have cost c_{ij}^2, and so on. It does not matter to which D-node the units of flow are directed. The flow via node c_i is bounded from above by $h_1 + \cdots + h_k$ as well as by the multiplicity N and from below by the multiplicity n.

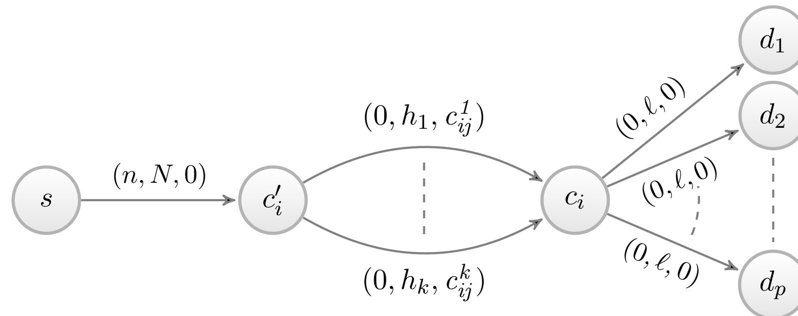


Figure 7.12: Several link priorities per object: links from an object c_i are associated with different costs.

Example 7.6. Suppose we have a switch with 20 ports and two network cables that are already connected to some of the ports (but lose on the other end). If we want to connect 4 computers to the switch, we will first use the two connected cables, as this causes no costs. As we need to connect two more computers to the switch, we have to buy further cables (or at least fetch them from the store, which costs time).

The flow network in Figure 7.12 is applied to this example as follows:

- $c_i \ldots$ switch
- $d_1 \ldots d_4 \ldots$ computers

- $N = 20 \ldots$ ports of the switch
- $n = 1 \ldots$ each switch has to be connected to at least one computer
- $m = M = 1 \ldots$ each computer has to be connected to exactly one switch
- $h_1 = 2 \ldots$ two connected network cables
- $h_2 = \ell - h_1 \ldots$ cables we have to buy (or fetch from the store)
  (in any case we need to buy at most ℓ cables, as this is the total flow in the network, i.e. the total number of cables to connect)
- $\ell \ldots$ number of links to distribute - we take the minimally required number of links, i.e. the total number of cables to connect; in this example we have $\ell = 4$

Figure 7.13 illustrates the corresponding flow network. Two units of flow can be transported from c_i to any D-node (i.e. from the switch to any computer) at cost c_{ij}^{1} (e.g. $c_{ij}^{1} = 0$) and the rest (at most $min(N-2, \ell-2)$ units of flow) at cost c_{ij}^{2} (e.g. $c_{ij}^{2} = 1$).

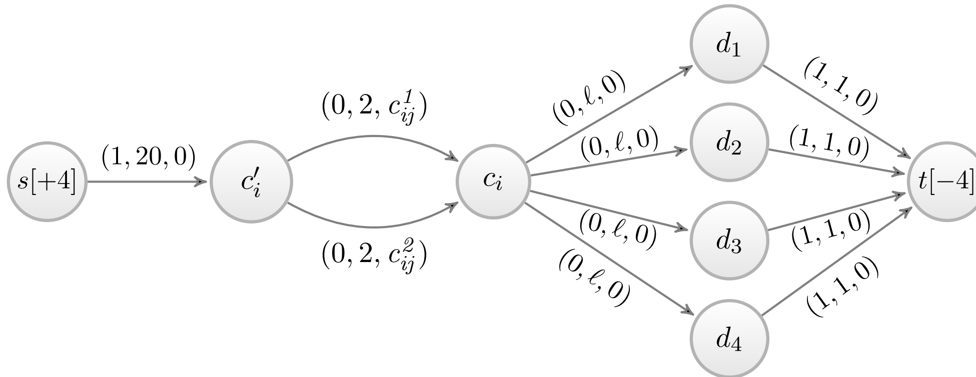


Figure 7.13: Links from switch c_i to computers $d_1 \ldots d_k$ can have different costs.

Object-pair-related Priority Links

Object-pair-related priority links, on the other hand, do not need additional network layers. We simply add one arc from c_i to d_j for each priority class of links between objects c_i and d_j. The network in Figure 7.14 distinguishes k different classes of links between particular objects c_i and d_j. The first h_1 units of flow from c_i to d_j can be transported at cost c_{ij}^{1}, the next h_2 units for cost c_{ij}^{2} and so on. If there are further D-nodes that have different classes of links to c_i, we connect them to c_i by multiple arcs as well and set the capacities and costs accordingly. All other D-nodes are connected to c_i with only one link at standard cost c_{std}. The same holds for d_j and other C-nodes. The flow from node c_i to node d_j is bounded as for object-related priority links.

One particular application of this approach is CONFIGURATIONREPAIR with *nonunique-nonunique* associations and existing links (for the general procedure see Section 7.5). For

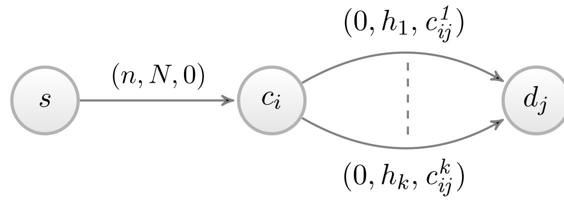


Figure 7.14: Several link priorities per pair of objects: links from object c_i to object d_j have different costs.

unique-unique associations, we only have one priority class per pair of objects, as each arc has a capacity of one. We therefore simply choose the cost of each arc accordingly. Hence we only need object-pair-related priority links for *nonunique-nonunique* associations (and only if there are more than one different priority links between some pairs of objects (c_i, d_j)).

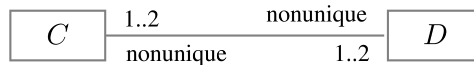


Figure 7.15: A *nonunique-nonunique* association.

Example 7.7. Consider the model in Figure 7.15 and a configuration with two objects per class and a total of four links. Suppose that neighbouring objects (with the same index) may be linked at cost 1, while other links have a cost of 2 (see Figure 7.16 for the corresponding flow network). This example illustrates why a uniform distribution of links does not always result in an optimal solution. In this case, a uniform distribution of links (connecting every C-object with every D-object) results in a total cost of 6, while the minimal cost of 4 is obtained by double links between neighbouring objects (i.e. a flow of 2 units from c_i to d_i for $i = 1, 2$), as shown in Figure 7.17.

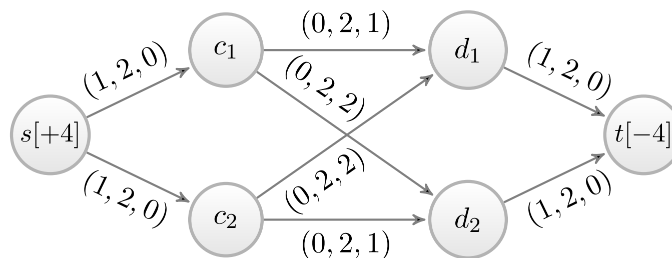


Figure 7.16: Flow network corresponding to an instance of the association in Figure 7.15 with two objects per class. The minimal cost of 4 is obtained for the flow $f(c_1, d_1) = f(c_2, d_2) = 2$ and $f(c_1, d_2) = f(c_2, d_1) = 0$.

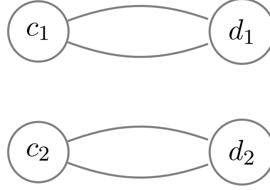
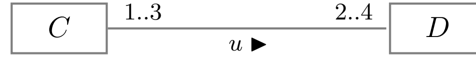


Figure 7.17: Satisfying instance of the association in Figure 7.15 with double links between c_1 and d_1 and between c_2 and d_2.

The following example illustrates how the CONFIGURATIONREPAIR problem can be solved for a *nonunique*-*nonunique* association.

Example 7.8. Consider a configuration $\mathfrak{I} = (\mathcal{O}, \mathcal{L})$ with two objects of class C and three objects of class D and some existing links: $\mathcal{O} = \{c_1, c_2, d_1, d_2, d_3\}$ and $\mathcal{L} = \{(c_1, d_1), (c_1, d_1), (c_2, d_3)\}$ (see Figure 7.19a). $\mathfrak{I}$ is an instance of the following *nonunique*-*nonunique* association:



The first link between objects c_2 and d_3 and the first two links between objects c_1 and d_1 have a cost of 0, all other links have a costs of 1. The corresponding network for finding a satisfying instance of the association is shown in Figure 7.18.

From the ILP-solver we get a range for the number of links: $[\ell_{\min}, \ell_{\max}] = [3, 8]$. As we want to maintain existing links, but still find a minimal solution, we take the minimum of $\ell_{\min}$ and $|\mathcal{L}|$ (the number of existing links). Hence we distribute $\ell = 3$ links. The link distribution corresponding to the solution of the minimum cost flow problem will either be $\mathcal{L}' = \{(c_1, d_1), (c_1, d_2), (c_2, d_3)\}$ (see Figure 7.19b) or $\mathcal{L}' = \{(c_1, d_1), (c_2, d_2), (c_2, d_3)\}$ with a total cost of 1. One existing link is removed, because otherwise the flow on the arc from d_2 to t would not be feasible (a flow of zero on an arc with a lower bound of one). Translated to the corresponding configuration the object d_2 would have no link to any C-object, thus violating the multiplicity 1..3.

By choosing $\ell = 4$ we can maintain all existing links and satisfy all D-objects: $\mathcal{L}' = \{(c_1, d_1), (c_1, d_1), (c_2, d_2), (c_2, d_3)\}$ (see Figure 7.19c). The total cost is the same as for $\ell = 3$.

Combinations of Priority Link Types

In scenarios where both object-related and object-pair-related priority links occur, we have to combine the two approaches in a common network.

Suppose there are k different link priorities for object c_i, leading to a flow network F like in Figure 7.12. Additionally, there are $x_{i,j}$ links between objects c_i and d_j. We add an arc from c_1' to d_1 with capacity x_{ij} and costs c_{ex} to the flow network F. The resulting network is shown in Figure 7.20.

If the object-related and the object-pair-related priorities occur at the same time, but for independent C-objects c_i and c_o, we use the procedure for object-related priority links for object

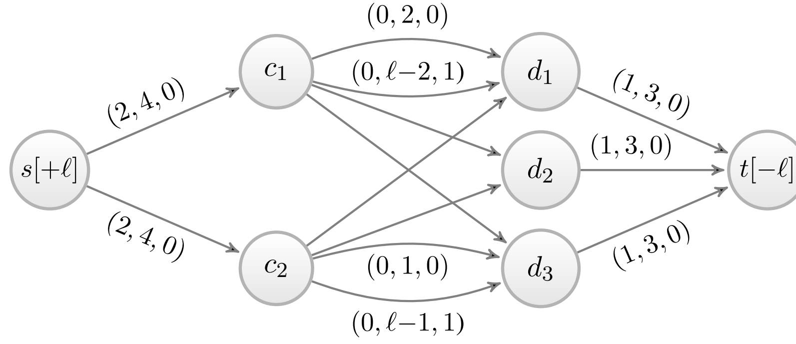
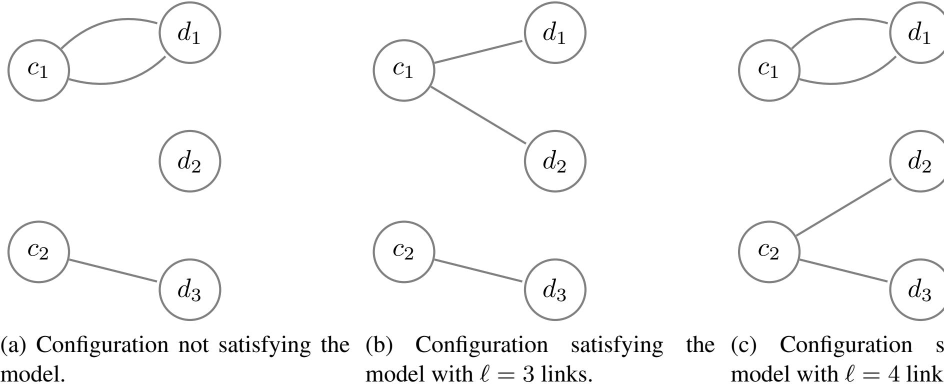


Figure 7.18: Flow network with links of different priority between specific pairs of objects. Unlabeled arcs have the following constraints: $(0, \ell, 1)$.



(a) Configuration not satisfying the model.

(b) Configuration satisfying the model with $\ell = 3$ links.

(c) Configuration satisfying the model with $\ell = 4$ links.

Figure 7.19: Instances of the model in Example 7.8.

c_i (i.e. add node c_i') and in addition the procedure for object-pair-related priority links for object c_o (i.e. add multiple links).

7.7 Choosing the number of links

The ILP-solver gives us an interval for the number of links that can be used to construct a configuration satisfying the given model: $\ell \in [\ell_{\min}, \ell_{\max}]$. To determine an appropriate number of links ℓ, we have to investigate the general structure of this problem.

For the LINKDISTRIBUTION problem, where we start from an empty set of links, we can choose any ℓ within the interval. In general, we are interested in minimising costs and therefore choose the minimal value $\ell_{\min}$.

When dealing with CONFIGURATIONCOMPLETION or CONFIGURATIONREPAIR, choosing the minimal ℓ might not result in the optimal solution. If we choose the minimum of $\ell_{\min}$ and the number of existing links $|\mathcal{L}|$ for ℓ, we might need to remove more existing links than if we chose

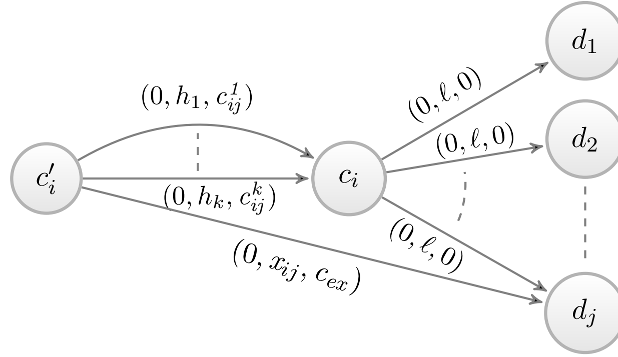


Figure 7.20: Links from c_i to D-objects have different costs, and there are x_{ij} existing links between c_i and d_j. (Nodes s and t omitted)

more links. We have seen in Example 7.8 that choosing $\ell = \ell_{\min} + 1$ results in a link distribution that maintains all links, whereas $\ell = \ell_{\min}$ leads to deleting one existing link. Depending on the costs of relinking and adding new links, choosing $\ell > \ell_{\min}$ might therefore be preferable.

In the case of CONFIGURATIONCOMPLETION (where we have to keep all existing links) there might not even be a solution with $\ell = \ell_{\min}$.

Example 7.9. Reconsider the configuration from Example 7.8. We cannot create a satisfying instance with $\ell = \ell_{\min} = 3$ without removing an existing link. We need to remove one link between c_1 and d_1 to be able to connect d_2 to either c_1 or c_2.

We can test whether an instance of a $nonunique$-$nonunique$ association is extendable at all by running a minimal cost flow algorithm with a supply of $\ell_{\max}$ units. If there is no solution with $\ell_{\max}$ links, the configuration is not extendable without removing any existing links.

Lemma 7.10. *Let $\mathfrak{M}$ be a model containing a nonunique-nonunique association, $\mathfrak{I}$ be a satisfying instance with ℓ links and $\ell_{\max}$ be the maximal possible number of links as computed by the* ILP*-solver ($\ell < \ell_{\max}$). Then we can always extend $\mathfrak{I}$ to a satisfying instance of $\mathfrak{M}$ with $\ell_{\max}$ links.*

Proof. The maximal number of links $\ell_{\max}$ of an instance of the association in Figure 7.2a is $\ell_{\max} = min(yM, xN)$, where x is the number of objects of class C and y the number of objects of class D.

If ℓ does not equal $\ell_{\max}$, there is still at least one C-object c_i and one D-object d_j that is not maximally linked. Therefore we can add a link between c_i and d_j. As we have a $nonunique$-$nonunique$ association, it does not matter if there already exists a link between c_i and d_j.

□

Lemma 7.10 implies that we can use a binary search to find the solution with the minimally required number of links, if there is a solution with $\ell_{\max}$ links.

For *unique-unique* associations, we cannot simply link two arbitrary objects c_i and d_j. If they are already linked, it is not possible to place an additional link between them. Therefore, we cannot always extend an instance of a *unique-unique* association to an instance with $\ell_{\max}$ links, as the following example shows. This means that in the *unique-unique* case there are repair problems that can be solved with less than $\ell_{\max}$ links, but not with $\ell_{\max}$ links,

Figure 7.21: Specification of a *unique-unique* association.

Example 7.11. Let $\mathfrak{M}$ be the model shown in Figure 7.21 and $\mathfrak{I} = (\mathcal{O}, \mathcal{L})$ an instance satisfying $\mathfrak{M}$ with $\mathcal{O} = \{a_1, a_2, a_3, b_1, b_2, b_3\}$ and $\mathcal{L} = \{(a_1, b_1), (a_1, b_2), (a_2, b_1), (a_2, b_2), (a_3, b_3)\}$ (see Figure 7.22a). The maximum number of links is $\ell_{\max} = 6$ and the current number of links is $\ell = 5$. We cannot add another link to $\mathfrak{I}$ without either violating the uniqueness constraint (by connecting a_3 and b_3 a second time) or violating some upper bounds (by connecting any of the other objects).

Figure 7.22b shows that a satisfying instance of $\mathfrak{M}$ with $\ell_{\max} = 6$ links indeed does exist. But this instance cannot be constructed by extending the configuration given in Figure 7.22a.

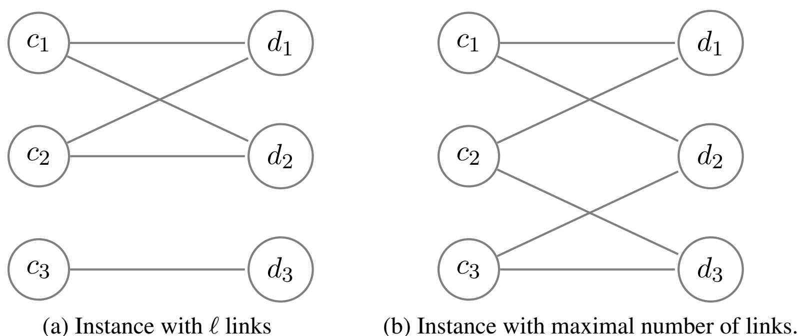


Figure 7.22: Instances of specification shown in Figure 7.21 with 3 objects of classes C and D

7.8 Different costs – different results

How we set the costs of existing and of new links depends on what the user wants to achieve. There are different ways of handling existing links when repairing configurations.

One possibility is to choose positive costs for both existing and new links, but with lower costs for existing links than for new links. In this case the minimum cost flow algorithm will prefer redirecting an existing link over maintaining all links and adding a new one.

Another possibility is to set the costs for existing links to zero and the costs for new links to a positive value. In this scenario keeping all links and adding a new link incurs the same costs as redirecting an existing link.

A third way of setting the costs is to choose negative costs for existing links (e.g. $c_{ex} = -1$) and positive costs for new links (e.g. $c_{std} > 0$). In this case it is more efficient to keep existing links and add new links than to redirect an existing link. This approach can result in a network where choosing more links results in a lower total cost, as we have negative costs for existing links. The following example illustrates this behaviour.

Example 7.12. Consider the configuration $\mathfrak{I}$ in Figure 7.23. It is an instance of the association in Figure 7.21. To extend $\mathfrak{I}$ to a satisfying instance of the association, we build the flow network shown in Figure 7.24 and solve a minimum cost flow problem.

For $\ell = 3$ we have to delete one of the links connected to c_1: either (c_1, d_1) or (c_1, d_2). Then we can add one link to d_3: either (c_1, d_3) or (c_2, d_3). Hence, one possible result of the MinCostFlow algorithm is: $f_{1,1} = 1$, $f_{1,3} = 1$, $f_{2,2} = 1$, with a total cost of -1. The resulting configuration can be seen in Figure 7.25a.

For $\ell = 4$, we can keep all existing links and just add a new link from c_2 to d_3. This results in a total cost of -2. The corresponding configuration is shown in Figure 7.25a. The configuration with four links is cheaper than the one with three links, hence we get more (links) for less (cost).

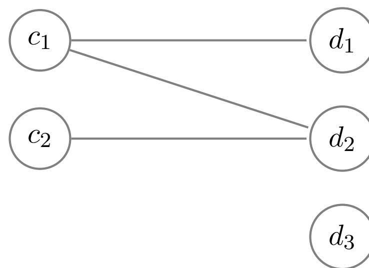


Figure 7.23: Nonsatisfying instance of the association shown in Figure 7.21 with 2 objects of class C and 3 objects of class D.

As we have negative costs for existing links, we can get more links for less cost. The next section investigates if this can also happen for non-negative costs.

7.9 Can we get more for less?

The More-for-less Paradox

In some situations something counter-intuitive happens: it is cheaper to transport more flow. This happens not only if we have negative cost on some arcs, but also if there are only positive costs. This is called the *more-for-less paradox* or *transportation paradox* and was first described by Charnes and Klingman [13] and Szwarc [41], but had already been noticed before.

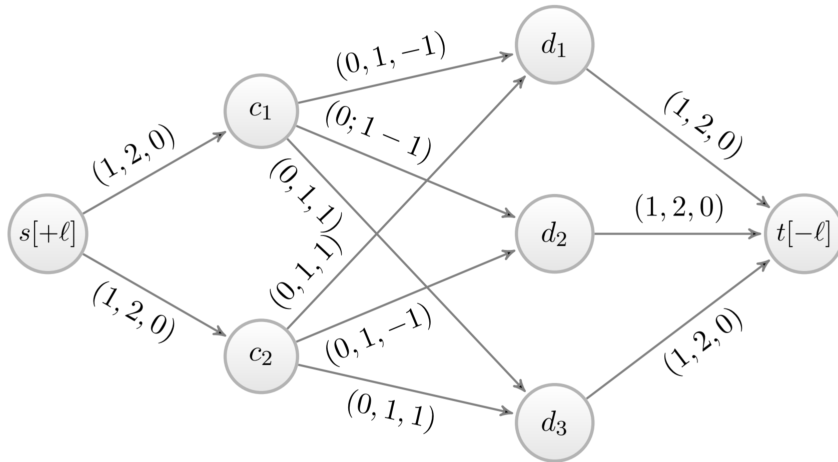
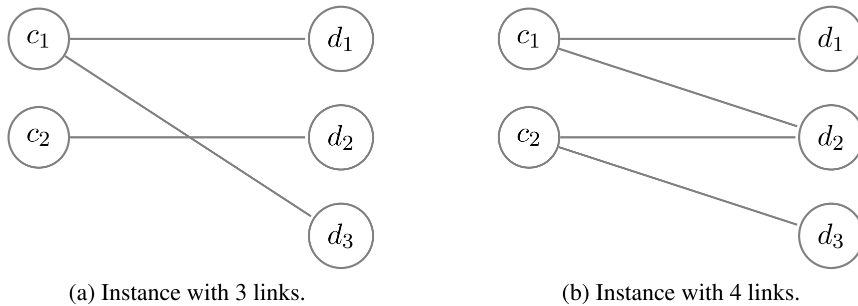


Figure 7.24: 4-layered graph to distribute links for the association shown in Figure 7.21 with 2 objects of class C and 3 objects of class D and existing links (see Figure 7.23).



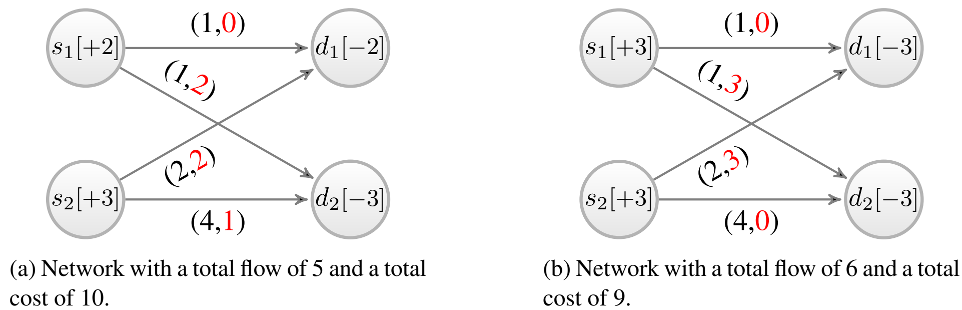
(a) Instance with 3 links.

(b) Instance with 4 links.

Figure 7.25: Satisfying instances of the specification shown in Figure 7.21

Charnes and Klingman defined the more-for-less paradox as follows: "given an optimal solution to a distribution problem, it is possible in certain instances to ship more total goods for less total cost even if we ship at least the same amount from each origin, and at least the same amount to each destination, and all the costs are non-negative" [13].

Example 7.13. Consider the transportation network given in Figure 7.26. Node s_1 has a supply of two units, node s_2 a supply of three units, node d_1 a demand of two units and d_2 a demand of three units. The optimal solution of the resulting transportation problem is shown in Figure 7.26a and has a total cost of 10. If we increase the supply at s_1 and the demand of d_2 by one, we get the transportation problem in Figure 7.26b with a total cost of 9. The total cost has decreased by one, although we send more goods (flow) through the transportation network.



(a) Network with a total flow of 5 and a total cost of 10.

(b) Network with a total flow of 6 and a total cost of 9.

Figure 7.26: Example of the more-for-less paradox for transportation problems. Numbers in square brackets give the supply (positive) or demand (negative) of a node, black numbers on arcs give the costs, red numbers the flow on the arc.

How about our flow networks?

The more-for-less paradox so far has only been described for networks with multiple sources and sinks. This paradoxical behaviour cannot happen with the networks we deal with, as we only have one source and one sink.

Proposition 7.14. *In a flow network with one source and one target and with only non-negative costs on each edge, the more-for-less paradox cannot occur.*

Proof. Consider all possible paths from the unique source to the unique target. Sending a unit of flow over any of these paths incurs a non-negative cost, which is the sum of costs for all edges on the path. Therefore the total cost of a flow increases monotonically with the flow. □

8 Related Work

> The skill of writing is to create a context in which other people can think.
>
> Edwin Schlossberg

This chapter gives an overview on related research in the areas of formalisation of UML Class Diagrams, configuration management, reconfiguration and redundancy detection.

8.1 Formalising UML Class Diagrams

As UML class diagrams play a central role in the design and specification of software, databases and ontologies, CASE (Computer Aided Software Engineering) tools should reveal syntax errors and provide feedback concerning errors, redundancies and inconsistencies [10]. To provide these additional features, the class diagrams have to be formalised. Formalising UML class diagrams is hence the basis for the methods presented in this thesis, as we introduce solutions to some of these challenges.

There are several approaches to translate the semantics of UML diagrams into a language suitable to perform formal reasoning; formal languages like Object-Z [31], Alloy [2], or description logic [11] have been used to reason about UML diagrams.

Choosing expressive formal languages to formalise the semantics of a UML class diagram has the advantage that these basic logics can handle different types of constraints. The specifications can hence contain all of these types of constraints simultaneously. For instance, the semantics of multiplicities specified in the class diagram and of constraints written in OCL can be expressed in the same first-order logic. Moreover, there exist well-developed reasoning techniques and theorem provers for these logics, which can be used for satisfiability and consistency checking. Calvanese et al. show in [12] that frame languages, semantic data models and object-oriented data models can be translated to a description logic called $\mathcal{ALUNI}$ and that satisfiability and subsumption of models can be checked in this framework.

The downside of these expressive, flexible languages is the high computational complexity of reasoning tasks. For example, checking the consistency of $\mathcal{ALUNI}$-specifications is EXPTIME-complete (see [7]). Recently *DL-Lite* [3,4] was introduced to address these complexity issues, with an emphasis on finite models [39].

Our approach of translating relevant fragments of class diagrams into an ILP formulation is based on the idea of obtaining polynomial-time algorithms for particular subsets of UML

class diagrams and OCL constraints. This technique was inspired by Lenzerini and Nobili who introduced a translation from ER diagrams to inequalities [32]. Feinerer and Salzer adapted this approach for UML class diagrams [17, 20] and extended and generalised the result [21], leading to a formal semantics for UML class diagrams (or at least of certain aspects thereof) and to polynomial procedures for reasoning tasks like consistency checking.

Balaban and Maraee [8] build on the approach by Lenzerini and Nobili as well and introduce algorithms for checking the (finite) satisfiability of UML class diagrams with class hierarchy constraints. They consider class hierarchies with generalisation constraints (overlapping/disjoint and complete/incomplete). They propose an efficient reduction of such diagrams to diagrams without class hierarchies, which replaces each hierarchy constraint by a linear number of additional associations. After reducing the diagram they can apply an algorithm introduced by Lenzerini and Nobili [32] to check the satisfiability of the diagram in polynomial time. The described procedure is called *FiniteSat*. In the presence of generalisation constraints they add an additional inequality for each constraint. The reduction as well as the introduction of the additional inequalities are linear in the size of the diagram. Moreover, Balaban et al. give a thorough discussion on reasoning with UML class diagrams in [10]. They give a summary of previous work on solving inconsistency and finite satisfiability and present a pattern-based approach for explaining and repairing correctness problems. The problems identified in this paper (which are inconsistencies, redundancies and abstraction errors) correspond to some of the problems found in the domain of configuration management by Falkner et al. [16].

In [9] Balaban and Maraee describe a translation from description logics (in particular ALC) to UML class diagrams. By running their *FiniteSat* algorithm on these class diagrams they are able to check the finite satisfiability of atomic, primitive knowledge bases of description logics.

Richters and Gogolla [38] support the UML design phase by validating UML models and OCL constraints. Validation, in contrast to verification, does not show the correctness of a model in a formal sense. It can nevertheless reveal constraints that are too strong or too weak by comparing system snapshots (i.e. simulated instances of the model) to the model. If either reasonable snapshots do not fulfil the constraints or unreasonable snapshots do fulfil them, the model has to be adapted.

Egyed [15] presents an approach for instant consistency checking for UML diagrams. He investigates sets of consistency rules and how changes need to be propagated for efficient evaluation. Contrary to our intention, he checks whether diagrams of different type are consistent to each other. Consider, for example, a class diagram specifying the structure of a system, a statechart diagram defining the behaviour of its classes and a sequence diagram describing processes within the system. Consistency rules are used to define conditions that have to be fulfilled by the models. The sequence diagram may e.g. only contain methods specified for the class invoking it (i.e. the class diagram has to contain the method in the respective class). Models violating these consistency rules are inconsistent. The presented method builds a scope for each rule, containing all model elements that affect its truth value. If a model element is changed, all rules containing this element in their scopes have to be revaluated. The presented method performed well on realistic examples in tests, requiring only a few milliseconds on average per model change.

8.2 Configuration Management and Reconfiguration

Domain experts often have difficulties using formal knowledge representation languages to specify product configuration systems. Felfernig, Friedrich et al. tackle this *knowledge acquisition bottleneck* in [24] by introducing UML configuration models. They add domain-specific modelling concepts with the help of stereotypes, the extension mechanism of UML, and define a mapping from these concepts to logical sentences, which can be transformed to different representations used by configuration tools. This mapping defines the semantics of the newly introduced concepts. The resulting *product models* (or *configuration models*) consist of classes, generalisation and aggregation (which are standard UML concepts) combined with *requires*- and *incompatible*-relations (which are domain-specific concepts) and OCL constraints. Additional customisations are ports (i.e. connection points) and connection relations, as well as resources that can be contributed or consumed by components. Both concepts are modelled by stereotyped classes and associations. The results are implemented in a prototype that uses the standard UML CASE-tool *Rational Rose*. The described approach leads to a significant reduction of time and costs for the development and maintenance of product configuration systems. In [26] Felfernig, Friedrich et al present translation rules from these UML configuration models to a corresponding OIL representation. With this approach they are able to provide a knowledge acquisition frontend based on UML thus facilitating the generation of configuration systems. Complex constraints that cannot be represented graphically are defined in languages such as OIL. The translation can be used to check the consistency of UML configuration models with the help of the reasoning support for Semantic Web ontology languages. The presented concepts are also implemented in a configuration knowledge acquisition workbench. In [25] the authors show how the UML can be employed to automatically construct configuration knowledge bases and how to diagnose and correct them with the help of positive and negative example configurations (i.e. examples that should be accepted resp. rejected by the knowledge base). They also discuss the diagnosis of inconsistent user requirements during product configuration (e.g. because feasible capabilities are exceeded). Additionally they describe an approach for reconfiguration, where they identify and remove elements of a configuration, such that the remaining parts of the configuration can be completed to a configuration satisfying all constraints.

As these papers show, the use of UML for modelling configuration problems is already established.

Aschinger et al. [5] have recently introduced LoCo, a declarative logical formalism for expressing configuration problems, which is a fragment of classical First Order Logic with existential counting quantifiers. They focus on describing configuration problems by a set of logical sentences. In this formalism the knowledge engineer only needs to specify the possible number of connections between two component kinds. Based on the translation from multiplicities to linear inequalities presented in [16] they infer finite bounds on the number of components. These bounds are needed to transform a problem model specified in LoCo into e.g. SAT. In [6] the language is extended and the authors discuss complexity results and present a prototypical implementation of LoCo.

The tasks of repairing existing configurations and the reconfiguration of established (legacy) systems have already a long tradition in knowledge-based configuration. In [40] Stumptner and

Wotawa present a model-based approach for reconfiguration. They use diagnosis mechanisms to find a set of configuration elements, such that altering this set yields a consistent configuration. This diagnosis process can provide an indication of where to start. The model on which the diagnosis is based contains an element for any entity that possesses a separate behaviour. Thus also an attribute may be a model element, if the system's correct behaviour depends on the value of the attribute. This approach does not only need a system description, but also observations about the component behaviour. The found diagnoses represent possible configurations that do not contradict the observations. From these configurations the ones providing the desired functionalities, so-called *suitable configurations*, are obtained by applying filter conditions. The two major factors identified for reconfiguration of an incorrect configuration are altered requirements and legacy systems.

Männistö et al. [33] introduce an abstract conceptual model for reconfiguration. In addition to the information needed for representing configuration knowledge they define a *reconfiguration model*, which consists of reconfiguration operations (containing a precondition that controls the applicability of the operation and an action) and reconfiguration invariants that specify correctness conditions for configurations. The authors formalise the reconfiguration process as a sequence of reconfiguration operations leading to a configuration that fulfils a set of conditions expressing new requirements. Additionally, they describe different modes of reconfiguration, ranging from non at all over project-based (i.e. individual) to automatic reconfiguration.

Friedrich et al. [27] present an approach for reconfiguration using answer set programming(ASP). They treat reconfiguration as an adapted configuration problem by specifying a reconfiguration problem as a set of (adapted) requirements, transformation rules and a legacy configuration to be reconfigured. Depending on the choice of modification costs, the problem solver will find different solutions. The authors also give several modelling patterns and evaluation results, which show that the ASPbased approach is feasible for an interesting set of reconfiguration problem instances.

We tackle the problem of reconfiguration in a different way by using an ILP solver and netflow algorithms. This approach only deals with multiplicity constraints, but has the advantage of providing a means for building configurations from scratch and for reconfiguration with the same approach.

8.3 Detecting Redundancies

Detecting redundancies was identified as a relevant problem that requires formal reasoning, not only in configuration management [16, 21] but also in UML class diagrams themselves [10].

Dullea and Song [14] analyse redundant relationships in entity relationship models. They argue that the fact that a relationship r is parallel to a composite relationship does not necessarily imply that r is redundant. An additional semantic constraint is necessary to ensure the redundancy. The authors consider minimum and maximum cardinality constraints for one-to-one, one-to-many, and many-to-many multiplicity types. They perform an exhaustive case study for combinations of these multiplicity types with a focus on binary associations. They investigate types of composite relationships that rule out redundancies (e.g. so-called fan relationships consisting of one relationship with maximum cardinalities of $M : 1$ and one with $1 : M$). Based

on this case study they develop a set of heuristics for two paths that are part of a cyclic path. They present several rules defining whether these relationship paths are redundant to each other or not.

Our work differs considerably from this approach since we investigate multiplicities specified by concrete intervals $[n..N]$, where both n and N may be any integers satisfying $N \geq n \geq 0$, instead of generic one-to-one (1:1), one-to-many (1:N), and many-to-many (M:N) multiplicities.

Even though equality constraints on association chains are a natural extension of class diagrams and often implicitly occur in the modelling phase, we are not aware of further research into this direction.

In the context of association chains checks for inconsistent or reducible multiplicities are of interest [22]. The composition of association chains can lead to multiplicity intervals where not every value of the interval can be reached. Hartmann discusses a formal representation of such multiplicities with gaps, so called int-cardinality constraints in [28] and also considers the consistency of these constraints. In [29] Hartmann considers the interaction of cardinality constraints with key and functional dependencies. This approach allows one to solve consistency and implication problems, but it does not seem to offer a method for tightening cardinalities.

9 Conclusion

Always and never are two words you should always remember never to use.

Wendell Johnson

In this thesis we presented solutions to some of the problems identified in the domain of configuration management. It summarises the results of our research of the past three years and gives more details on the underlying theory of the main achievements.

We gave an in-depth analysis of the number of objects of some class D connected to a number of objects of another class C via a specific association. Based on these results we developed formulas for calculating the multiplicities of a composed association. This means that we view a chain of k associations from an initial class C_0 to a final class C_k as one composed association from C_0 to C_k.

We then extended UML class diagrams by equations over association chains, so-called *equality constraints*, to express additional properties of relations instantiating associations and association chains. We used these equality constraints to detect redundant multiplicity bounds, i.e. values that are not reachable due to the multiplicity constraints imposed by the parallel association chain. We described how to derive tighter bounds for individual multiplicities and developed an algorithm to reduce all multiplicities of a given model, thus leading to a *reduced model*. The results have partly been implemented in a prototype that is available from [35]. The system has the flavour of a spreadsheet program, since it re-checks the consistency of specifications and configurations as well as the redundancy of multiplicities with every change, highlighting inconsistencies and redundancies as an immediate feedback. Additionally, we have implemented a Prolog prototype [18] to conduct further experiments. This thesis focused on $unique\text{-}unique$ associations only, but results on composing $nonunique\text{-}nonunique$ associations have been published in [22]. Reducing multiplicities for $nonunique\text{-}nonunique$ still remains an open issue. For this purpose we first need to investigate the meaning of equations over $nonunique\text{-}nonunique$ association (chains).

In the context of composed associations we also detected the existence of multiplicity intervals, where not every value of the interval can be realised. We showed that our approach of equating associations and reducing multiplicities can detect the unsatisfiability of models where the intervals on both sides of the equation have no common values. We have not yet investigated approaches to find gaps in intervals that do have common values. Combining Hartmann's ap-

proach of int-cardinality constraints (see [28]) with our approach of composing associations and dealing with equality constraints seems to be a promising extension.

Moreover, we discussed the effects of equality constraints concerning satisfiability and minimal instances. Based on examples we demonstrated that the minimal E-satisfying instance of a model is not always identical to the minimal E-satisfying instance. In general the latter will be larger. We introduced a special family of models that is characterised by so-called *tree-generating* equations. For these models we were able to derive an explicit formula for the reduction of the upper multiplicity bound. We also showed that the E-satisfiability can be checked using the inequalities we get from the model for specific subcases of this family. Unfortunately the minimal instance and the minimal E-satisfying instance are not even identical for tree-generating models. Further research has to be done to find out how much larger an E-satisfying instance can get in the worst case. Our next aim is to search for a (dis)proof of the conjecture that E-satisfiability can be reduced to satisfiability by our method.

Furthermore, we introduced a method for distributing links between objects, such that the link distribution satisfies a given model. We use minimum cost flow algorithms to accomplish this task. This approach can as well be used to solve the problem of reconfiguration, which is a central challenge when dealing with long-lived component systems. We also investigated, how the choice of costs in the flow network influences the reconfiguration result. Nevertheless we still need to continue the research in this area. Another important open issue is how to proceed when the repair process fails. Our aim is to obtain information about bottlenecks from the failed reconfiguration attempts and to develop a semi-automated procedure that includes the user in the reconfiguration process. We also have not yet found a solution for (re-)distributing links in the presence of equality constraints. Another question to be investigated is whether our approach (especially the method for reconfiguration) can be integrated with more general approaches developed for the task of reconfiguration of product configuration systems.

With our numerical approach we could already build a framework for solving several problems. By translating UML class diagrams into linear inequalities and solving the resulting ILP problem with a solver we can check the consistency and satisfiability of a model and check whether instances satisfy a model. Additionally the ILP solver calculates the number of objects of each class needed to form a minimal instance and gives a range for the number of links for each association. The links can then be distributed between the objects with the help of netflow algorithms as described in this thesis. As most ILP solvers can solve netflow algorithms, we can generate a complete configuration using only one tool. Reconfiguration (i.e. adding and deleting objects from a given configuration and rearranging links) can also be solved via netflow algorithms. Furthermore, we can compute the type of composed associations and find minimal multiplicities based on the inequalities.

As a next step we will investigate further classes of constraints as seen in large-scale applications of our industrial projects partners which can be efficiently handled within our framework. For example we are planning on integrating the reduction from diagrams containing ISA constraints to diagrams without such constraints presented by Balaban and Maraee [8] with our approach to be able to handle ISA constraints as well.

Apart from the few enhancements already described above, we currently think that the limits of this numerical approach are (nearly) reached. Using expressive formal languages (e.g. first

order logics) on the other hand poses other limitations, as most problems in first-order logic are undecidable. This means that currently mainly user-guided semi-automated procedures exist, with the users having to be trained in the underlying theory since they have to interfere with the system when it gets stuck. As our numerical approach allows for efficient (automated) reasoning about and optimisation of UML class diagrams, while translating models to symbolic logic frequently leads to algorithms of high complexity – a complexity that often is not inherent in the original problem but is introduced by using (too) expressive languages, our vision is to develop a hybrid system. Basic constraints can thus be handled by our approach and the results can be used as a starting point for approaches with a higher computational complexity. One step into this direction has already been done by Aschinger et al. [5], who build on our translation into inequalities to infer bounds on the number of components. The bounds are e.g. needed to formulate a SAT problem. Hybrid systems can thus benefit from the advantages of both worlds and limit their downsides. We take advantage of the efficiency of numerical methods for the limited range of basic UML elements and constraints that these methods are suited for. Further elements and constraints can then be handled by other formal languages, trading in low complexity of reasoning tasks for the high expressiveness of those formalisms.

In [18] Feinerer (who is also part of our research team) argued about the benefits and drawbacks of declarative formalisms like Prolog for configuration systems. To solve the problems the author proposes hybrid systems, combining the effective strategy of "generate and test" with external tools specialised on specific tasks like instance generation. These parts from the external tools can then be combined to a configuration, which can be tested for validity by the declarative framework. One of the next aims clearly is to further extend this approach (and the corresponding Prolog implementation as well, as long as this seems sensible) to build a powerful framework for configuration management.

Bibliography

[1] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows – theory, algorithms and applications*. Prentice Hall, 1993.

[2] Kyriakos Anastasakis, Behzad Bordbar, Geri Georg, and Indrakshi Ray. On challenges of model transformation from UML to Alloy. *Software and System Modeling*, 9(1):69–86, 2010.

[3] Alessandro Artale, Diego Calvanese, Roman Kontchakov, Vladislav Ryzhikov, and Michael Zakharyaschev. Reasoning over extended ER models. In Christine Parent and et al., editors, *Proc. Conceptual Modeling ER 2007*, volume 4801 of *LNCS*, pages 277–292. Springer, 2007.

[4] Alessandro Artale, Diego Calvanese, Roman Kontchakov, and Michael Zakharyaschev. Adding weight to DL-Lite. In Bernardo Cuenca Grau and et al., editors, *DL 2009*, volume 477 of *CEUR Workshop*, 2008.

[5] Markus Aschinger, Conrad Drescher, and Georg Gottlob. Introducing LoCo, a logic for configuration problems. In *Proceedings of LoCoCo 2011*, Perugia, Italy, 2011.

[6] Markus Aschinger, Conrad Drescher, and Heribert Vollmer. Loco – a logic for configuration problems. In *Proceedings of the 20th European Conference on Artificial Intelligence, ECAI 2012*, 2012.

[7] Franz Baader and et al., editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[8] Mira Balaban and Azzam Maraee. Consistency of UML class diagrams with hierarchy constraints. In *Proc. NGITS2006*, volume 4032 of *LNCS*, pages 71–82. Springer, 2006.

[9] Mira Balaban and Azzam Maraee. A uml-based method for deciding finite satisfiability in description logics. In *Description Logics*, 2008.

[10] Mira Balaban, Azzam Maraee, and Arnon Sturm. Management of correctness problems in uml class diagrams towards a pattern-based approach. *IJISMD*, 1(4):24–47, 2010.

[11] Daniela Berardi, Diego Calvanese, and Giuseppe De Giacomo. Reasoning on UML class diagrams. *Artificial Intelligence*, 168(1–2):70–118, 2005.

[12] Diego Calvanese, Maurizio Lenzerini, and Daniele Nardi. Unifying class-based representation formalisms. *Journal of Artificial Intelligence Research*, 11:199–240, 1999.

[13] A. Charnes and D. Klingman. The more for less paradox in the distribution model. *Cahiers du Centre d'Etudes de Recherche Operationelle*, 13(1):11–22, 1971.

[14] James Dullea and Il-Yeol Song. An analysis of cardinality constraints in redundant relationships. In *Proceedings of CIKM '97*, pages 270–277. ACM, 1997.

[15] Alexander Egyed. Instant consistency checking for the UML. In *ICSE '06: Proceeding of the 28th international conference on Software engineering*, pages 381–390, New York, NY, USA, 2006. ACM Press.

[16] Andreas Falkner, Ingo Feinerer, Gernot Salzer, and Gottfried Schenner. Computing product configurations via UML and integer linear programming. *Int. J. Mass Cust.*, 3(4), 2010.

[17] Ingo Feinerer. *A Formal Treatment of UML Class Diagrams as an Efficient Method for Configuration Management*. Dissertation, Vienna University of Technology, 2007.

[18] Ingo Feinerer. Towards hybrid techniques for efficient declarative configuration. In Wolfgang Mayer and Patrick Albert, editors, *Proc. ECAI2012 Workshop on Configuration*, pages 21–24, 2012.

[19] Ingo Feinerer, Gerhard Niederbrucker, Gernot Salzer, and Tanja Sisel. Configuration repair via flow networks. In Li Chen, Alexander Felfernig, Jiming Liu, and Zbigniew W. Ras, editors, *Foundations of Intelligent Systems – 20th International Symposium, ISMIS 2012, Macau, China, December 4–7, 2012. Proceedings*, volume 7661 of *Lecture Notes in Computer Science*, pages 321–330. Springer, 2012.

[20] Ingo Feinerer and Gernot Salzer. Consistency and minimality of UML class specifications with multiplicities and uniqueness constraints. In *Proceedings of the 1st IEEE/IFIP International Symposium on Theoretical Aspects of Software Engineering, June 6–8, 2007, Shanghai, China*, pages 411–420. IEEE Computer Society Press, 2007.

[21] Ingo Feinerer and Gernot Salzer. Numeric semantics of class diagrams with multiplicity and uniqueness constraints. *Software and Systems Modeling*, 2013. To appear.

[22] Ingo Feinerer, Gernot Salzer, and Tanja Sisel. Reducing multiplicities in class diagrams. In Jon Whittle, Tony Clark, and Thomas Kühne, editors, *Model Driven Engineering Languages and Systems, 14th International Conference, MODELS 2011, Wellington, New Zealand, October 16–21, 2011. Proceedings*, volume 6981 of *Lecture Notes in Computer Science*, pages 379–393. Springer-Verlag, October 2011.

[23] Ingo Feinerer, Gernot Salzer, and Tanja Sisel. Class diagrams with equated association chains. In *7th International Symposium on Theoretical Aspects of Software Engineering, TASE 2013, July 1-3, 2013, Birmingham, UK*. IEEE Computer Society, 2013. To appear.

[24] Alexander Felfernig, Gerhard Friedrich, and Dietmar Jannach. UML as domain specific language for the construction of knowledge-based configuration systems. *International Journal of Software Engineering and Knowledge Engineering*, 10(4):449–469, 2000.

[25] Alexander Felfernig, Gerhard Friedrich, and Dietmar Jannach. Conceptual modeling for configuration of mass-customizable products. *AI in Engineering*, 15(2):165–176, 2001.

[26] Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach, Markus Stumptner, and Markus Zanker. Uml as knowledge acquisition frontend for semantic web configuration knowledge bases. In Michael Schroeder and Gerd Wagner, editors, *RuleML*, volume 60 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2002.

[27] Gerhard Friedrich, Anna Ryabokon, Andreas A. Falkner, Alois Haselböck, Gottfried Schenner, and Herwig Schreiner. (re)configuration based on model generation. In Conrad Drescher, Inês Lynce, and Ralf Treinen, editors, *Proceedings of LoCoCo 2011*, volume 65 of *EPTCS*, pages 26–35, 2011.

[28] Sven Hartmann. On the consistency of int-cardinality constraints. In Tok Wang Ling and et al., editors, *Conceptual Modeling - ER '98*, volume 1507 of *LNCS*, pages 150–163, 1998.

[29] Sven Hartmann. On interactions of cardinality constraints, key, and functional dependencies. In Klaus-Dieter Schewe and Bernhard Thalheim, editors, *Proceedings of FoIKS2000*, volume 1762 of *LNCS*, pages 136–155. Springer, 2000.

[30] M. Hitz, G. Kappel, E. Kapsammer, and W. Retschitzegger. *UML @ Work, Objektorientierte Modellierung mit UML 2*. dpunkt.verlag, 3. edition, 2005 (in German).

[31] Soon-Kyeong Kim and David A. Carrington. Formalizing the UML class diagram using Object-Z. In Robert B. France and et al., editors, *Proc. UML'99*, volume 1723 of *LNCS*, pages 83–98, 1999.

[32] Maurizio Lenzerini and Paolo Nobili. On the satisfiability of dependency constraints in entity-relationship schemata. *Information Systems*, 15(4):453–461, 1990.

[33] Tomi Männistö, Timo Soininen, Juha Tiihonen, and Reijo Sulonen. Framework and conceptual model for reconfiguration. Technical report, AAAI Conf. Workshop, AAAI Press, 1999.

[34] Gerhard Niederbrucker. A numeric semantics for UML class diagrams: Methods and tools. Master's thesis, Technische Universität Wien, 2010.

[35] Gerhard Niederbrucker and Tanja Sisel. *Clews Website*, 2011. `http://www.logic.at/clews`.

[36] Object Management Group. *Unified Modeling Language 2.4.1*, 2011. `www.omg.org/spec/UML/2.4.1/`.

[37] Object Management Group. *Object Constraint Language 2.3.1*, 2012. `www.omg.org/spec/OCL/2.3.1/`.

[38] Mark Richters and Martin Gogolla. Validating UML models and OCL constraints. In Andy Evans, Stuart Kent, and Bran Selic, editors, *UML 2000 - The Unified Modeling Language. Advancing the Standard. Third International Conference, York, UK, October 2000, Proceedings*, volume 1939 of *LNCS*, pages 265–277. Springer, 2000.

[39] Riccardo Rosati. Finite model reasoning in DL-Lite. In Sean Bechhofer and et al., editors, *Proceedings of ESWC2008*, volume 5021 of *LNCS*, pages 215–229. Springer, 2008.

[40] Markus Stumptner and Franz Wotawa. Model-based reconfiguration. In *In Proceedings Artificial Intelligence in Design*, pages 45–64. Kluwer Academic Publishers, 1998.

[41] W. Szwarc. The transportation paradox. *Naval Res. Logist. Quarterly*, 18(1):185–202, 1971.

Curriculum Vitae

Personal Data

Name:	Tanja Sisel
Date of Birth:	August 27th 1981, Vienna
Address:	Schrekergasse 46/4/12 1160 Wien Österreich
E-Mail:	tenza@gmx.at

Education

September 1987 – June 1993	Friedrich Eymann Waldorfschule, 1130 Wien (Primary and Secondary School)
September 1993 – June 1995	BRG 17 Parhammerplatz, 1170 Wien (Secondary School)
September 1995 – June 2000	ORG für Leistungssport Maroltingergasse, 1160 Wien (5-jährig, Sportart: Eiskunstlauf) (Secondary School for competitive sports – figure skating)
June 2000	**Matura** mit Auszeichnung (Graduation with distinction)
Sept. 2000 – Feb. 2003	Vienna University of Technology (TU Wien) Informatik und Wirtschaftsinformatik, Diplomstudien (Studies of Computer Science and Business Informatics)
March 2003 – Sept. 2005	Vienna University of Technology (TU Wien) **Bakkalaurea technicae (B.Sc.)** in Medicine and Computer Science ("Medizinische Informatik")
October 2005 – July 2009	Vienna University of Technology (TU Wien) **Diplom-Ingenieurin (M.Sc.)** in Medicine and Computer Science ("Medizinische Informatik") with distinction (mit Auszeichnung)
September 2009 - Now	Vienna University of Technology (TU Wien) Doktoratsstudium der technischen Wissenschaften, Informatik (PhD-Studies, Computer Science)

Additional Education and grants

August 1999	Cannes, Frankreich **"Diplôm de Langue Francaise"**
2009	**Forschungsstipendium** der TU Wien
April 2011	Poster-presentation at conference **"Einsteins in the City"**, New York, funded by TU Wien

Publications and Theses

Tanja Sisel. Development of a Glycan-Binding Protein Database-Platform.
Master's thesis, Vienna University of Technology, 2009

Ingo Feinerer, Gernot Salzer, and Tanja Sisel. Reducing multiplicities in class diagrams. In Jon Whittle, Tony Clark, and Thomas Kühne, editors, Model Driven Engineering Languages and Systems, 14th International Conference, MODELS 2011, Wellington, New Zealand, October 16–21, 2011. Proceedings, volume 6981 of Lecture Notes in Computer Science, pages 379–393. Springer-Verlag, October 2011.

Ingo Feinerer, Gerhard Niederbrucker, Gernot Salzer, and Tanja Sisel. Configuration repair via flow networks. In Li Chen, Alexander Felfernig, Jiming Liu, and Zbigniew W. Ras, editors, Foundations of Intelligent Systems – 20th International Symposium, ISMIS 2012, Macau, China, December 4–7, 2012. Proceedings, volume 7661 of Lecture Notes in Computer Science, pages 321–330. Springer, 2012.

Ingo Feinerer, Gernot Salzer, and Tanja Sisel. Class diagrams with equated association chains. In 7th International Symposium on Theoretical Aspects of Software Engineering, TASE 2013, July 1-3, 2013, Birmingham, UK. IEEE Computer Society, 2013.

Talks

„Reducing multiplicities in class diagrams", MODELS 2011 Conference, Wellington (Neuseeland), Oktober 2011

„Configuration repair via flow networks", ISMIS 2012 Conference, Macao (China), Dez. 2012

Professional Experience

February 2001 – April 2010	Fritsch, Chiari & Partner (FCP), Wien Part-time employment, IT department (Development of database-applications with MS Access)
April 2010 – March 2013	TU Wien, Universitäts-Assistentin (Predoc), Institute for Computer Languages, Theory and Logic Group
April 2013 – May 2013	TU Wien, Project Assistant, Institute for Computer Languages, Theory and Logic Group

Main projects

Database-application for the financial management of a project (for the "ÖBB")

Database-application for the administration and evaluation of "Telecom-Austria"-invoices

Database- application for the valuation of structural damages of residential buildings (for "Wiener Wohnen")

Several applications for internal usage (e.g. password-management)

Private Activities

Competitive Figure Skating (1988-1999)

Formation dancing (latin-american dances, 1999-2007)

Volunteer at "Dancer Against Cancer" (charity ball for "Österreichische Krebshilfe") since 2007