Terminological knowledge representation systems in a process engineering application

Von der Mathematisch-Naturwissenschaftlichen Fakultät der
Rheinisch-Westfälischen Technischen Hochschule Aachen
zur Erlangung des akademischen Grades einer
Doktorin der Naturwissenschaften
genehmigte Dissertation

vorgelegt von
Diplom Informatikerin Ulrike Sattler
aus München

Referent: Universitätsprofessor Dr.-Ing. F. Baader
Korreferent: Universitätsprofessor Dr.-Ing. W. Marquardt
Tag der mündlichen Prüfung: 29. Mai 1998

Contents

I would like to thank all those who helped me start, set up, and complete this thesis—by providing scientific and technical support and guidance, coffee, laughter, friendship, dart games, well-being, optimism, fun, fuzz, and love.

Chapter 1

Introduction

This thesis is concerned with the question of how far terminological knowledge representation sytems can support the development of mathematical models of chemical processes. In this chapter, after a very brief description of the goals of process systems engineering, it will be argued that the *structuring* of the application domain is crucial for the development of process models. Then the main ideas and features of terminological knowledge representation sytems are sketched, and it is argued that the system services provided by a terminological knowledge representation system can support the structuring of an application domain. Furthermore, the reasons why the process systems engineering application asks for the extension of the expressive power of the formalism underlying terminological knowledge respresentation systems, so-called *Description Logics*, will be discussed.

1.1 Process Systems Engineering

Process systems engineering is concerned with the construction of models of chemical processes. We can think of these processes as the ones taking place in possibly huge chemical plants, even though process systems engineering is not restricted to those. With this image in mind, it should be clear that trial and error methods for the design and optimisation of these processes are not only too costly and time-consuming, but also too dangerous. Furthermore, as the competition in the market of chemical products is very strong, the developers of chemical processes aim at a high level of optimisation. This does not only concern the quality and quantity of the process' product (that is the substance the process is intended to produce), but also the resources

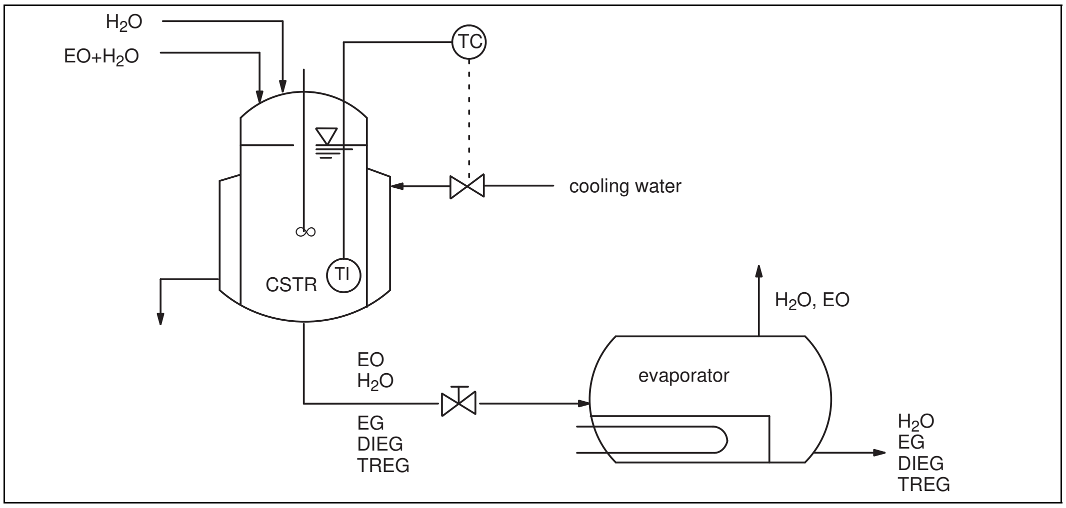

Figure 1.1: Flowsheet of the ethylene-glycol process.

the process needs for its production, the amount and composition of other substances produced by the process, the level of safety, etc. Hence, instead of trial and error methods, process engineers construct *models* whose purpose it is to enable the (numerical) simulation, analysis, and understanding of chemical processes.

Models of chemical processes come in different shapes according to the specific requirements of the model and the state of its development: They can be natural language descriptions of the process, miniature plants in a laboratory, flow sheets (graphics using some standard notations for reactors, pipes, valves, thermometers, etc.) and other graphical models, or sets of equations describing the behaviour of the process. For example, a flowsheet representation of an ethylene-glycol process can be found in Figure 1.1. It consists mainly of a reactor—which is equipped with a stirring unit and a cooling jacket—and an evaporator, and these two parts are connected via a pipe equipped with a valve. In Figure 1.2, the same process can be found in the ModKit representation, which is very close to the one used within VeDa. A graphical representation in VeDa notation of the reactor in the ethylene-glycol process can be found in Figure 1.3. In this representation, the connection between the reactor, or, more precisely, the cooling jacket and the environment is emphasised. Finally, the decomposition of the gas phase material balance equation describing the behaviour of the evaporator can be found in Figure 1.4. It is denoted using a so-called *and-or* graph in order to represent both the parts an equation consists of (using *and* nodes), and alternatives one has for the description of a single term in the equation (using *or* nodes).

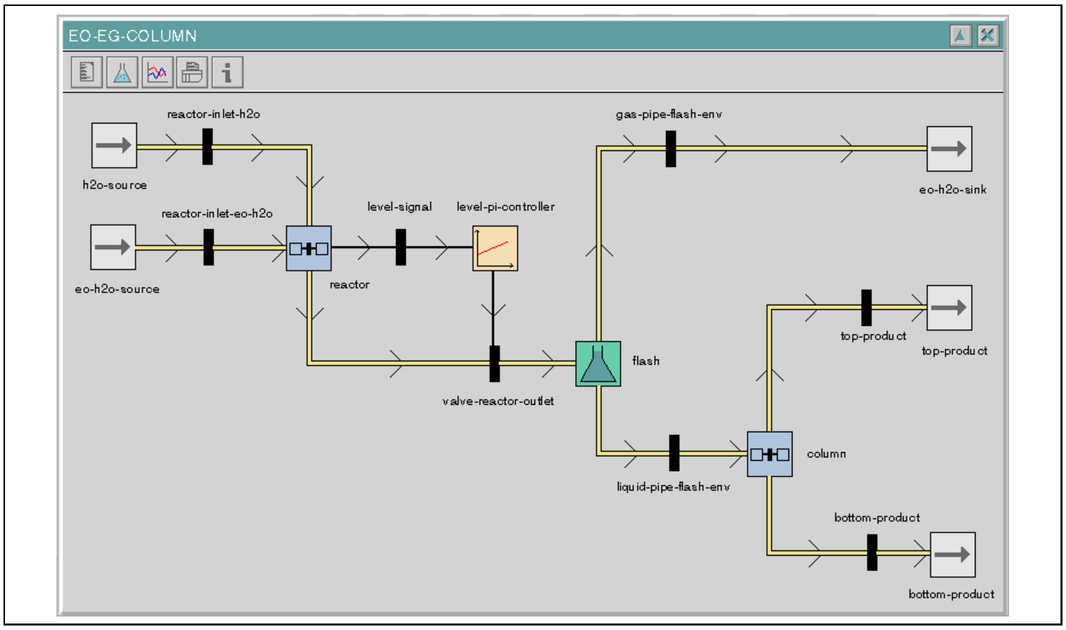

Figure 1.2: The ethylene-glycol process in ModKit representation.

Models can also be classified according to the goal they are designed to achieve. For example, there are models designed for the precise simulation of the behaviour of the process, models designed in order to get a better understanding of the way in which the process functions and in which the parts of this process interact, and there are decision support models, whose goal it is to support decisions that are to be taken while working on an appropriate solution for the design problem.

Mathematical models, namely sets of differential, integral, or algebraic equations and mixtures of these equations, are of a special interest since they are the prerequisites for the analysis and simulation of the behaviour of a process. A mathematical model consists, in general, of a huge number of equations (hundreds or thousands of equations, depending on the process and the degree of detailing of the process model) and it can thus hardly be solved exactly—neither by hand nor by a symbolic solver—but only approximately by a numerical solver or simulation tool. To use such a tool, it is necessary to transform the mathematical model into the input format of this tool. Then the engineer can simulate the influence which particular parameters, like the inlet temperature to the reactor, have on the outcome of the process. Other tools allow the optimisation of particular process parameters, like the time the process needs for the production of the target substance or the ratio between

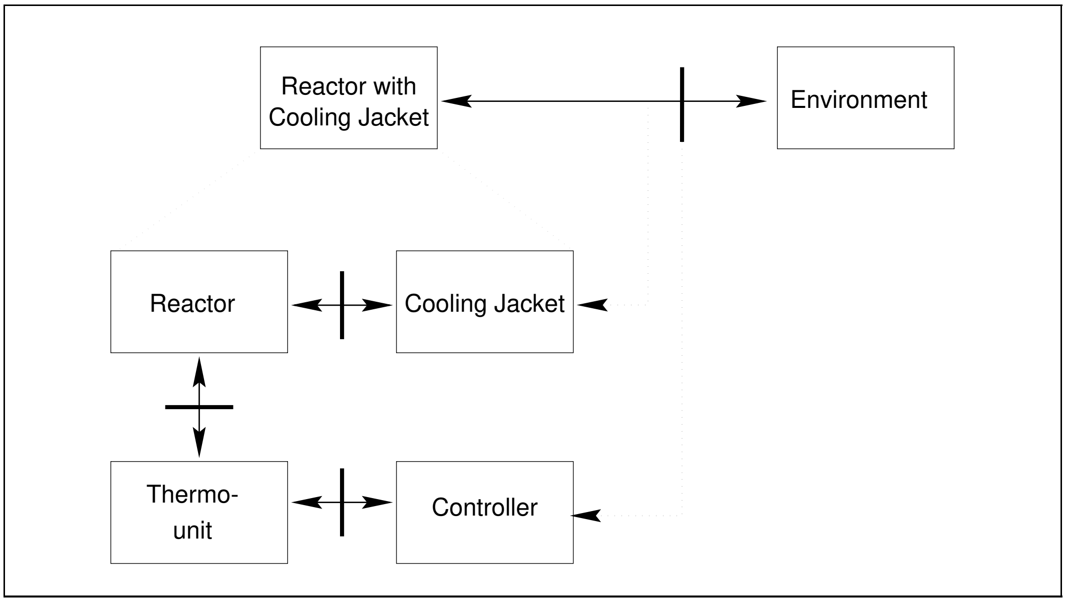

Figure 1.3: Decomposition of the reactor of the ethylene-glycol process into building blocks.

the target and the input substances.

There are mainly three factors contributing to the complexity of the construction of a precise mathematical model:

1. The possibly very high complexity of a process together with the great variety of phenomena that can take place within a process.
2. The fact that the more precisely a mathematical model describes the behaviour of a process, the more equations are contained in it.
3. The difficulties human beings have in the perception of facts and interrelationships that are denoted in mathematical formulae.

Thus, in general, different graphical models serve as outlines for the construction of a mathematical model.

Summing up, we are confronted with various transitions from one representation formalism to another: From one graphical formalism to another one, from a graphical formalism to a mathematical one, and from a mathematical one into a language that is understood by a numerical solver or simulation tool.

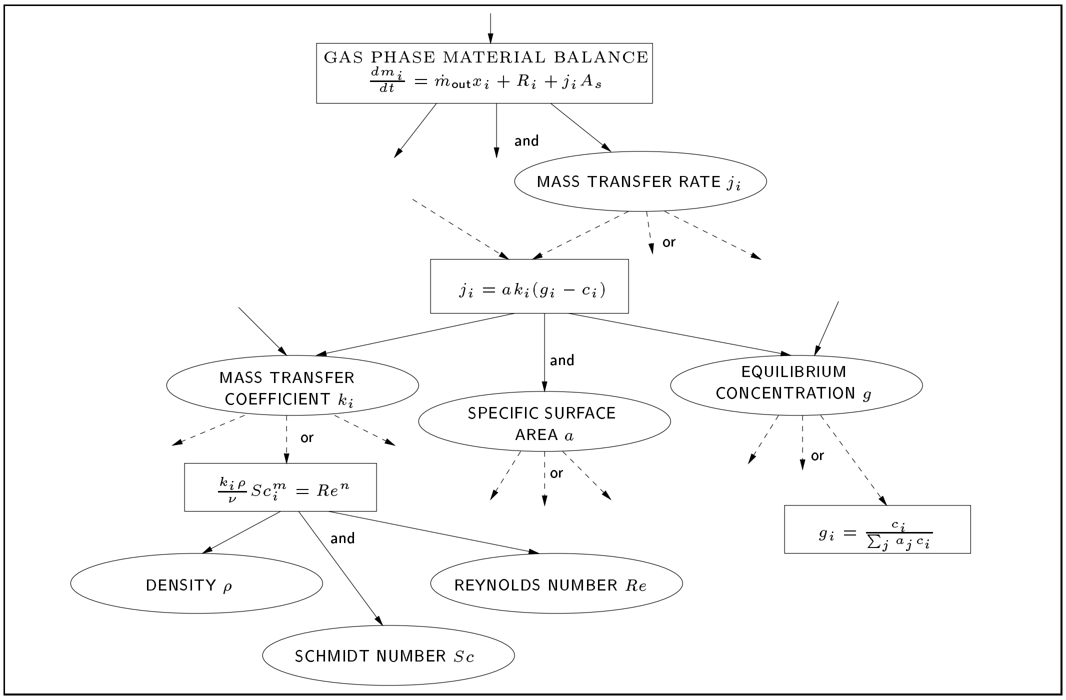

Figure 1.4: Decomposition of the gas phase material balance equation.

Unfortunately, there is no software tool available that supports these transitions between representation formalisms. Furthermore, there is neither an agreed methodology for the development of process models nor a description of the objects these models can be viewed to consist of [Marquardt1996]. Without going into detail here, these objects will be called (basic) building blocks and can be thought of as devices, connections, or descriptions of the behaviour of devices or connections. Using these building blocks, a model could be characterised by a description of the building blocks it consists of together with a description of how these buildings blocks are interrelated. Until now, these building blocks and their properties strongly depended on the representation formalism used for a particular kind of model. As a consequence, during the life-cycle of a process model, one has to build several models using several formalisms. First, one starts with natural language models, then moves to various graphical formalisms, then designs a mathematical model and/or writes the source code for a numerical solver or simulation tool. It is not possible to use one single model while increasing its degree of detailing or possibly changing the graphical representation of its building blocks according to the goal and state of the model. This fact not only increases the cost of modeling chemical processes, but it also makes the reuse of already developed models more difficult and is an additional source of errors.

To overcome this shortage, W. Marquardt and co-workers at the Lehrstuhl für Prozeßtechnik of RWTH Aachen (Aachen University of Technology) are developing such a description of the standard building blocks, the so-called VeDa ("Verfahrenstechnisches Datenmodell", german for "chemical engineering data model"). It consists of three parts: (1) a frame-based formalism for the representation of standard building blocks, (2) the description of these standard building blocks within this formalism, and (3) the description of modeling steps, i.e., the steps an engineer undertakes while building a process model. VeDa is a step towards a model development environment supporting the above mentioned transitions from one representation formalism to another one: The standard building blocks are independent from the graphical representation; they are generic in that they comprise all objects a process model might comprise, and they are flexible in that they can be changed according to the specific requirements of a model.

Since the number of standard building blocks is relatively large, they must be stored in a structured way—otherwise, the reuse of already defined building blocks becomes difficult because it is hard to find the ones one is looking for. In VeDa, objects are grouped into classes, and classes are ordered with respect to the inheritance relation which should, according to the intended semantics of VeDa, coincide with the is-a-specialisation-of relation. That is to say that if a class B inherited some of its properties from a class A, then B should also be more specific than A. Now, due to the strong expressive power of VeDa, this cannot always be guaranteed, and the consequence of this fact is shown in the following example.

Example: A user of VeDa could define a class `reactor` using inheritance from a class `tank` and specify its properties by mistake in such a way that an instance of `reactor` is not necessarily also an instance of `tank`. As a consequence, the user (and possibly other users as well) might get confused since the structure he or she thinks the classes have is different from the one they actually have. One can think of a situation where an engineer is searching the instances of `tank`, expecting to find a certain t in it. Now, it is possible that this t belongs to the "sub"-class `reactor` of `tank` and not to `tank`, and that the engineer will not find t because he or she was looking for t in the "wrong" place.

In order to avoid misunderstandings and to really profit from the structured storage, it should always be kept consistent with the way the structure is defined. More precisely, if classes are structured with respect to the is-a-specialisation-of relation, then the system should guarantee that a class A

which is said to be more specific than a class B is really more specific than B—that is, each object belonging to A also belongs to B.

The number of building blocks is continuously growing, hence the structure should also be flexible and allow for changes according to new building blocks. This means that it should be possible to add a new, appropriate class for objects that do not fit well into already existing ones, or to divide a class A into subclasses if A has become too unspecific because of the growing number of objects in it.

To guarantee the consistency of this structure and to help the engineers in using and extending it, automatic reasoning services could be used. Whether these automatic reasoning services are realizable or not depends only on the formalism which is used to describe these building blocks: This formalism has to allow for the computation of these reasoning services. For example, these reasoning services and the above mentioned flexibility are not provided by standard data base systems—which have other features—but they are provided by some knowledge representation systems. In fact, the family of knowledge representation systems based on Description Logics are equipped with a powerful language to describe interesting properties of the objects in an application domain and they provide the reasoning services necessary to guarantee the consistency of the structured storage of these objects. Furthermore, for a variety of these knowledge representation systems, questions concerning the expressive power and the computational complexity of the according reasoning services are already formulated and investigated.

In a nutshell, the motivation behind the cooperation between the developers of VeDa at the Lehrstuhl für Prozeßtechnik and the Lehr- und Forschungsgebiet Theoretische Informatik was the question: "Which representation formalism has enough expressive power for describing the relevant properties of the building blocks of process models *and* allows for the automatic computation of the reasoning services needed to guarantee consistency of their structured storage?" The first results obtained within this cooperation are presented in this thesis.

1.2 Terminological Knowledge Representation Systems and Description Logics

Terminological Knowledge Representation Systems are based upon *Description Logics*, a formalism also known as Concept Languages, KL-One-based knowledge representation languages, or Terminological Logics.

After a brief introduction of Description Logics, it is argued that terminological knowledge representation systems (DL systems) are good candidates for tools supporting the structured storage of building blocks.

Description Logics are a family of logic-based representation formalisms whose origin can be found in *semantic networks*. Beside the pure formalism, there are various implementations of knowledge representation systems that are based on Description Logics [Patel-Schneider *et al.*1991; MacGregor1991; Baader *et al.* 1994; Bresciani *et al.*1995; Horrocks & Rector1996]. Starting in the late eighties, a large variety of different Description Logics has been defined and investigated, almost all of whom have the following properties in common:

- They are equipped with precise, well-defined semantics. This makes the behaviour of the knowledge representation system independent of a particular implementation and enables the precise definition of inference problems and reasoning services.

- In most but not all cases, they allow only the use of unary predicates (such as `Device` or `Connection`) and binary predicates (such as `connected-to` or `has-part`).

- System services of DL systems deduce *implicit* knowledge from the one *explicitly* given by the user. Most Description Logics are built such that they have *decidable* inference problems (see, for example, [Hopcroft & Ullman1997] for an introduction to the theory of complexity and computability). Hence the system services of a DL system are effectively computable (even if most of them are of a relatively high computational complexity). For an extensive overview on reasoning problems and techniques see [Donini *et al.*1996].

These properties, the last one in particular, lead to the assumption that Description Logics are a good starting point for searching a representation formalism as described at the end of Section 1.1.

A Description Logic is mainly characterised by a set of *constructors* that can be used to build complex *concepts* and *roles* from atomic concepts (unary predicates) and roles (binary predicates). Throughout this thesis, to distinguish more easily between concepts and roles, the latter start with lower case letters whereas concepts start with upper case letters. For example, devices which are only connected to pipes and have a cooling jacket can be described by the following concept:[1]

$$\texttt{Device} \sqcap (\forall \texttt{connected-to.Pipe}) \sqcap (\exists \texttt{has-part.Cooling-jacket}).$$

Devices having no cooling jacket as parts and at least 3 parts can be described by

$$\texttt{Device} \sqcap (\forall \texttt{has-part}.\neg \texttt{Cooling-jacket}) \sqcap (\geq 3\ \texttt{has-part}).$$

Using these constructors, the *terminology* of an application domain can then be defined in a so-called *TBox* by a set of concept- and role-definitions, as for example the following two concept definitions describing cooled reactors and cooled reactors equipped with a stirring unit:

$$\begin{aligned} \texttt{CoolReactor} &:= \texttt{Reactor} \sqcap (\exists \texttt{has-part.Cooling-Jacket}), \\ \texttt{StCoolReactor} &:= \texttt{CoolReactor} \sqcap (\exists \texttt{has-part.Stirring-Unit}), \end{aligned}$$

where we assume that the concepts occurring in these definitions are already defined. Referring to these concepts, concrete objects of a concrete "world" are described in a so-called *ABox*. An example of which being:

$$\begin{gathered} \texttt{REACTOR2} : \texttt{Reactor}, \\ \texttt{REACTOR2 has-part COOL17}, \\ \texttt{COOL17} : \texttt{Cooling-Jacket}, \end{gathered}$$

which states that the object `REACTOR2` is an instance of `Reactor` and related via the role `has-part` to the object `COOL17`, which is a `Cooling-Jacket`. Given the above concept definitions, a knowledge representation system should then be able to infer that `REACTOR2` is a `CoolReactor`. This inference—classifying objects with respect to the concepts defined in a TBox—is one of the system services that are provided by DL systems. Other system services comprise, for example,

- the computation of the implicit *subsumption* relation[2] between two concepts, or, more generally, the computation of the *taxonomy*, i.e. the hierarchy of concepts defined in a TBox with respect to the subsumption relation,
- deciding whether a given concept is *satisfiable*, i.e. whether it can ever be instantiated,

[1] Precise definitions of syntax and semantics of Description Logics are given in Section 4.2; an overview can be found in the Appendix.

[2] This relation is defined in Definition 4.2. Roughly spoken, a concept D subsumes a concept C if each instance of C is always also an instance of D.

- *classification* of objects, i.e., enumerating the most specific concepts a given object is an instance of, or

- *retrieving* all objects occurring in an ABox that are instances of a given concept.

Given that objects are grouped into concepts that correspond to classes, the subsumption relation on concepts is equivalent to the is-a-specialisation-of relation on classes mentioned in Section 1.1. Thus, the automatic computation of the subsumption relation and the automatic classification of objects are the reasoning services available in DL systems which can be used to guarantee the consistency of the structured storage of objects.

Since the late eighties, numerous investigations concerning the expressive power and the computational properties of various Description Logics have been carried out [Donini *et al.*1991a; 1991b]. These investigation were often motivated by the use of certain constructors in implemented systems [Nebel1988; Borgida & Patel-Schneider1994], or the need for these constructors in specific applications [Baader & Hanschke1993; Franconi1994; Sattler1996] and the results influenced the design of new systems [Horrocks1997].

An important constructor for concepts in Description Logics are the so-called *number-restrictions*. These allow to describe objects by restricting the number of objects which are related to them by some role. For example, ($\geq$ 4 `has-part`) describes all those objects having at least 4 parts, i.e., those objects having at least 4 objects related to them via the role `has-part`.

There are two main reasons why number restrictions are present in almost all implemented systems. Firstly, human beings tend to describe objects using number restrictions. Secondly, number restrictions can express information concerning the way in which objects are interrelated by roles. This information on the relation between objects is of great importance for a process systems engineering application because process models strongly depend on the way in which their buildings blocks are interrelated.

Unfortunately, most Description Logics only allow the use of *atomic* roles inside number restrictions—whereas *complex* roles are excluded from being used in number restrictions. Complex roles are built using role-forming constructors such as union, intersection, inverse and composition. For example, these constructors can be used to describe the role `connected-to` as the union of `input-connected-to` and `output-connected-to`, and the role `is-part-of` as the inverse of `has-part`.

This restriction to atomic roles is unsatisfactory since (1) we must distinguish roles that are also allowed inside number restrictions from those that can be used in other places of concepts but not inside number restrictions. This makes the syntax of Description Logics more complicated and more difficult to learn for the user, and (2) the expressive power of number restrictions increases if complex roles built by the above mentioned role constructors can be used inside them. Using composition of roles, for example, one could describe a device such that all devices connected to it are controlled by the same control unit by:

$$\texttt{Device} \sqcap (\leq 1\ \texttt{connected-to} \circ \texttt{controlled-by}). \tag{1.1}$$

The number restriction in (1.1) states that each of these devices is related to at most one object via the role `connected-to` followed by `controlled-by`. To ensure that the device itself is also controlled by the unit that controls the devices connected to it, we additionally need union in the number restriction:

$$\texttt{Device} \sqcap (\leq 1\ \texttt{controlled-by} \sqcup \texttt{connected-to} \circ \texttt{controlled-by}). \tag{1.2}$$

Inversion of roles comes in if we need the role `controls` as well. An important part of this thesis is concerned with the influence complex roles in number restrictions have on the computational complexity of the corresponding inference problems.

Another weakness of traditional number restrictions comes up if we try to describe devices having fewer inputs than outputs (referred to by `Output-devs` in the following) or devices whose parts have the same number of inputs and outputs (referred to by `Equal-parts-devs`). If an upper bound n on the number of inputs can be fixed, `Output-devs` can be described by the following disjunction:

$$\begin{aligned}\texttt{Device}\ \sqcap\ (&((= 0\ \texttt{input}) \sqcap (\geq 1\ \texttt{output})) \sqcup \\ &((= 1\ \texttt{input}) \sqcap (\geq 2\ \texttt{output})) \sqcup \\ &((= 2\ \texttt{input}) \sqcap (\geq 3\ \texttt{output})) \sqcup \\ &\quad \vdots \quad \vdots \\ &((= n\ \texttt{input}) \sqcap (\geq n+1\ \texttt{output}))).\end{aligned}$$

If we cannot fix this upper bound, it is not obvious how `Output-devs` can be described since infinite disjunction is not allowed in Description Logics.

To overcome this lack of expressiveness, we introduce numerical variables $\alpha, \beta, \ldots$ to be used in number restrictions. Thus, `Output-devs` can be described by

$$\texttt{Device} \sqcap (= \alpha\ \texttt{input}) \sqcap (> \alpha\ \texttt{output}), \tag{1.3}$$

where α stands for some nonnegative integer. Similarly, `Equal-parts-devs` can be described by

$$\texttt{Device} \sqcap (\forall \texttt{has-part}.((= \alpha\ \texttt{input}) \sqcap (= \alpha\ \texttt{output}))). \tag{1.4}$$

This concept reveals a certain ambiguity: It describes devices where each part has the same number of inputs and outputs. However, it depends on the reading whether different parts can have different numbers of inputs or not. To overcome this ambiguity, we introduced explicit existential quantification of numerical variables (denoted by $\downarrow\alpha$) to distinguish between (1.5) a device where for each of its parts the number of its inputs equals the number of its outputs,

$$\texttt{Device} \sqcap (\forall \texttt{has-part}\,.\,(\downarrow\alpha.(= \alpha\ \texttt{input}) \sqcap (= \alpha\ \texttt{output}))), \tag{1.5}$$

and (1.6) a device where, for some number n, all parts have n inputs and n outputs,

$$\texttt{Device} \sqcap (\downarrow\alpha.(\forall \texttt{has-part}\,.\,(= \alpha\ \texttt{input}) \sqcap (= \alpha\ \texttt{output}))). \tag{1.6}$$

The investigation of these so-called *symbolic* number restrictions forms another important part of this thesis. Finally, this application asks for the adequate representation of composite objects: The high complexity of most process models asks for the possibility to model a process with different degrees of detailing by decomposing devices into their parts or aggregating parts of a model to a device or to a connection. Hence the relation between parts and the whole they belong to, the *part-whole relation*, plays an important rôle[3] for the adequate representation of process models. Beside its other properties, the part-whole relation is transitive. Thus, the adequate representation of complex objects asks for a Description Logics that allows for transitive roles. A Description Logic has been presented in [Baader1991] that allows the transitive closure of roles inside concepts. Using this role-forming constructor, it is possible to distinguish between objects having a (direct) part that is a valve connection,

$$\texttt{Device} \sqcap (\exists \texttt{has-part}.\texttt{Valve-Connection}),$$

and objects having at some (arbitrary but finite) level of decomposition a valve connection,

$$\texttt{Device} \sqcap (\exists \texttt{has-part}^{+}.\texttt{Valve-Connection}),$$

where $\texttt{has-part}^{+}$ denotes the transitive closure of the role `has-part`. The reasoning services of DL systems allowing for the transitive closure constructor are

[3]In this thesis, the spelling “rôle” is used to refer to roles in general in order to distinguish them from “roles” in the context of Description Logics, where they denote binary relations.

still decidable, and there exist decision procedures for these reasoning services. However, this constructor strongly increases the computational complexity of these services. Looking for a possibility to evade this high complexity, other ways of extending Description Logics by transitive roles were investigated. The results of this investigation form another important part of this thesis.

The above mentioned part-whole relation has several subrelations, such as the one between a whole and its ingredients or the one between a whole and its components. Intuitively, these subrelations have different properties and interact in specific ways. Hence they are also of importance for the appropriate representation of complex objects. In the literature, several characterisations of these subrelations can be found [Simons1987; Winston *et al.*1987; Gerstl & Pribbenow1993; Pribbenow1995], but as none of these characterisations was precise enough to decide which are the relevant part-whole relations for the process systems engineering application, an investigation of these sub-part-whole relations is also described within this thesis.

Besides these theoretical investigations, the assumption that DL systems are appropriate for supporting the structured storage of process model building blocks was tested empirically: The DL system Crack [Bresciani *et al.*1995] was integrated into the process modeling tool ModKit [Bogusch *et al.*1996]. ModKit is developed under the real-time expert system shell G2 [Gen1995], which is a powerful tool that lacks, however, means for the structured storage of objects. As a consequence, in ModKit, classes of standard building blocks were only poorly structured, and Crack was integrated to overcome this shortcoming. A description of this integration as well as a résumé of the experience we had with this integration can be found in Chapter 5.

1.3 Structure of the Thesis

This thesis was written while I was a member of Postgraduate College "Informatics and Technology" (Graduiertenkolleg "Informatik und Technik") at the RWTH Aachen. The aim of this Graduiertenkolleg is to establish a close cooperation between computer science and technological applications in order to both identify applications for new computer science techniques and to work on and solve complex problems in engineering applications. Hence, this thesis addresses both, engineers and computer scientists. I tried to write most parts of all chapters except Chapters 6 and 7 in a way that they can also be read by someone who is not a computer scientist. The computer science readers are asked to apologise for the remarks which might be, in their eyes, unnecessary and redundant, and which were made for readers who are not computer scientists.

The rest of this thesis is organised as follows:

- Chapter 2 discusses the problems one encounters while constructing models of chemical processes and sketches the process modeling methodology developed within VeDa.

- In Chapter 3, a framework for the classification of part-whole relations is presented. These relations play an important rôle for the representation of aggregated object and had to be understood before designing a representation formalism for process systems engineering.

- Chapter 4 contains an introduction to Description Logics in order to make the thesis self-contained. Besides the definition of syntax and semantics of the Description Logic the future investigations and extensions are based on, it includes a sketch of how Description Logics are used within a knowledge representation system and the idea of the basic inference algorithm. Finally, it presents already existing extensions of this basic Description Logic together with a discussion of their expressive power.

- In Chapter 5, the integration of a DL system into a process modeling tool is described. This integration was realised during my time at the Postgraduate College and motivated by the wish to learn more about the usefulness of DL systems from concrete scenarios, especially more about the way in which the powerful inference services of these systems can be used within such a process modeling tool.

- Chapter 6 describes the extensions of the basic Description Logics by so-called expressive number restrictions. These extensions increase the expressive power of Description Logics in a way that is useful for the process systems engineering application. The first extension allows the user to restrict the number of objects an object is related to via complex relations. The second extension allows the user to state, for example, that an object has the same number of inputs and output—without being more precise on this number. Both extensions are motivated by the process systems engineering application and investigated with respect to the computational complexity of the relevant inference problems.

- In Chapter 7, three extensions of the basic Description Logic by transitive relations are investigated. These extension are mainly motivated by the need for an appropriate representation of aggregated objects. All three extensions have decidable inference problems, and it is proved that one is of a much lower computational complexity than the others. Then the consequences of these results for the representation of aggregated objects are described.

- Finally, Chapter 8 contains the conclusion and summary of the thesis. After this chapter, the appendix contains a short summary of the syntax and semantics of the Description Logics used within this thesis.

Chapter 2

Process Modelling

In this chapter, we describe the process systems engineering application and the problems one encounters when modeling chemical processes. The development of these models is a rather creative and complex task which is described in the first part of this chapter. Methodologies of how to develop such a model evolved only recently. One of these methodologies was developed within VeDa [Marquardt1996], and its main ideas are briefly sketched in the second part of this chapter.

2.1 Modeling chemical processes

Process systems engineering is concerned with setting up models of chemical processes. Mostly, these processes take place in chemical plants that are run by chemical industries for the production of chemical substances.

More precisely, a *process* is an amalgamation of physical, chemical, biological, and informational events, which are undertaken in order to change substances with respect to their nature, properties, and composition. A *model* is a goal-driven simplification of the reality obtained by abstraction. Hence a *process model* is a model of a process, and its goal is to enable the simulation, analysis, and understanding of a given process. This understanding is the prerequisite for improving already developed processes and for designing new, high quality processes.

As stated in Section 1.1, setting up a process model of some precision is a complex task. This is mainly due to the following peculiarities of process systems engineering:

- There is a great variety of different chemical devices and connections (such as reactors, pipes, valves, distillation columns, etc.) which have to be considered on an abstract level.

- There is a great variety of different physical phenomena (such as heat transfer, reaction, evaporation, dispersion, mixing, etc.) which also have to be considered on an abstract level.

- Abstracting from a physico-chemical phenomenon is nondeterministic: Many phenomena can be viewed from different angles and thus abstracted by different means, and their behaviour can be approximated by various different heuristics.

- The requirements with respect to the granularity and the precision of the models is increasing with the state-of-the-art, the competition in the market, and the standards concerning environmental and safety demands.

The overall number of different models is infinite because the size of a model (i.e., the number of the devices and connections it contains) is not bounded. This large number of models would be of purely theoretical interest, but the number of different processes that are described by these models is also very large. Moreover, even one particular process taking place in a specific plant can be modeled in many different ways—which differ not only in their degrees of detailing, but also in the way phenomena are abstracted. This freedom of choice contributes tremendously to the complexity of building a process model. Various software tools which support the development of process models help the engineers to accomplish this task, increase the efficiency of the modeled processes, and contribute to an improvement of the security of chemical plants.

Typically, the final process model that is to be built is a *mathematical* model, i.e., a set of differential, integral, and algebraic equations as well as mixtures of these types of equations. Depending on the degree of detailing and the complexity of the process, such a set can contain several thousands of equations. The route that is taken to reach this model can be viewed as taking place in a three dimensional space [Pohl1995; Jarke & Marquardt1995]:

1. The *specification* dimension relates to the degree of understanding the engineers have of the process and thus to the degree of detailing of the model. In general, the engineers start the modeling at the plant level. The model is then refined, possibly down to the level of physico-chemical phenomena at the molecular level.

2. The *representation* dimension relates to the representation formalism that is used to describe the current model. In the early stages, the process can be described in a natural language specification. Then semi-formal graphical representation formalisms can be used for a more precise description of the process. Next, the informal graphical formalism can be replaced by a more formal graphical representation formalism. Finally, the behaviour of the process is described by a mathematical model.

3. Following the *agreement* dimension, the degree of agreement among the different engineers involved in the development of a process model should increase while setting up this model.

In general, process models can be viewed as being built from so-called *building blocks*, which are the entities process models consist of. In the VeDa methodology, the central building blocks are devices, connections, entities describing the behaviour of devices and connections, and entities describing the interfaces between devices and connections. By *standard* building blocks, we mean generic building blocks commonly used. The following is a possible scenario of the development of a process model:

1. The engineer chooses standard building blocks such as devices, connections, etc., from a catalogue. This is a well-structured collection of building blocks together with descriptions of their properties and possibly equipped with powerful search facilities.

2. If necessary, these building blocks are modified according to the requirements of the current model. These modifications can include both a more detailed specification of properties that are unspecified so far as well as complete changes of the properties of a building block.

3. The building blocks are connected to each other, possibly using their already specified interfaces.

4. The building blocks are broken down into their components until an appropriate degree of detailing is reached.

5. The behaviour of the building blocks is specified. First, for each building block, the phenomena that are associated to it are fixed. Then process quantities (i.e., variables standing for relevant process quantities) are introduced for the basic phenomena. Equations relate these variables to each other and might refine some of them. Finally, if necessary, initial values are fixed for process variables.

6. The information specified so far is used to automatically generate the source code for a numerical solver which describes the behaviour of this process.

7. If it is likely that parts of this model can be reused in other models, they are stored as new building blocks in the catalogue.

To assist the engineer, each of these steps should be controlled by a mechanism that guarantees the consistency of the model and the modifications that are carried out by these steps. This means, for example, that adaptations of the standard building blocks to the requirements of the current model (see Step 2) are compared with the actual descriptions of the building blocks in order to guarantee consistency of the model. If, for example, the engineer chooses from the catalogue a building block that represents a stirred-reactor and then removes the stirring unit, the system should notice that this building block does no longer represent a stirred-reactor but a reactor.

Unfortunately, no modeling tool supporting the above scenario is available. In fact, current modeling tools for chemical processes can be divided into two groups [Marquardt1996]:

- Modeling using an *equation-oriented* tool (e.g., SPEEDUP or DIVA) consists of either the implementation of a mathematical model using a declarative programming language together with a model library, or its implementation using a procedural programming language and a set of subroutine templates. These tools are very flexible in that they allow for a large variety of different models. The price one has to pay for this flexibility is the high effort required for modeling .

- Modeling using a *block-oriented* tool (e.g., ASPEN PLUS) takes place on the flow-sheet level, where the engineer selects, specifies and connects standard building blocks, so-called process units, from a model library using either a modeling language or a graphical editor. A severe restriction of these tools is due to the fact that standard building blocks can only be modified in a restricted way to match the specific needs of a modeling context. Hence, the degree of detailing of the models one can construct using these tools is bounded, which restricts the applicability of these tools. As a recompense, they are rather comfortable to use.

Even if these different tools could be integrated into one general modeling tool, points 2, 4, 6, and 7 of the above mentioned scenario would not automatically be supported: The block oriented tools are restricted to the predefined building blocks which cannot be broken down arbitrarily. Furthermore, a methodology is still lacking which describes the behaviour of devices or connections in such a precise way that the according set of mathematical equations and the complying source code for numerical solvers can be generated automatically. First attempts at the automatic generation of source code from models developed using an object-oriented modeling tool were made within ModKit, a modeling tool partially described in Chapter 5.

In order to realise the above scenario, the new building blocks have to be stored in a way that enables the user to find a building block he or she is looking for. This means that they must be arranged with respect to some meaningful order—otherwise, it would be hard to find the building blocks one wants to reuse. In the following, we will refer to a storage of building blocks with respect to such a meaningful order by *structured storage*.

The more difficult it is to find the building blocks one is looking for, the less one will try to reuse already specified blocks. Hence the structured storage of building blocks will lead to a higher degree of efficiency in the development of process models.

As a consequence of the above observations, the need for a representation formalism which can be used for both the description of the structural and behavioural aspects of building blocks arises. Additionally, this formalism should be able to support the structured storage of building blocks. A promising candidate for the corresponding meaningful order is the "is-a-specialisation-of" relation. More precisely, one unites similar building blocks into *classes* and associates a description of the properties of its members to each class. Then the resulting set of classes (and thus their instances) can be arranged with respect to the "is-a-specialisation-of" relation. In order to avoid inconsistencies, this specialisation relation should not be hand-coded by the engineers, but it should be computed automatically using the descriptions of the classes. Furthermore, class definitions should be tested automatically for inconsistencies or contradictions. In order to enable the automatic computation of the specialisation relation and the consistency test, the properties of a class (i.e., the properties of the objects belonging to this class) should be described in a declarative way using a formalism with well-defined semantics. Furthermore, the problem of whether one class is a specialisation of another class, or whether a class is inconsistent, should be effectively decidable. In Chapter 4, a family of knowledge representation formalisms having these properties, the so-called Description Logics, is introduced. The integration of a knowledge representation system based on Description Logics into a modeling tool for chemical processes is then described in Chapter 5.

2.2 VeDa, a chemical engineering data model

VeDa is short for "*Verfahrenstechnisches Datenmodel*" (chemical engineering data model). It was first developed by W. Marquardt [Marquardt1992]. VeDa is a frame-based formalism (see [Baumeister1998] for a description of the underlying formalism) designed for the formal representation of the modeling methodology developed at the Lehrstuhl für Prozeßtechnik of RWTH Aachen. Both the formalism and the methodology are described in [Marquardt1996]. The main characteristics of this methodology are the following.

VeDa distinguishes two aspects of a process model: Its *structural* and its *behavioural* aspects. This distinction is useful because of the following freedom of choice: Firstly, the structure of a model is not uniquely determined by the specific process it models and the degree of detailing of the model. Secondly, having fixed the structure of a model, its behaviour can be mostly modeled in various different ways. This is due to the fact that both the modeling of the behaviour and of the structure strongly depend on the goal and the precision of the model. For example, the behaviour of a stirring reactor can be modeled, in one model, by describing the kinetics of the reaction taking place in this reactor. In this case, the equations describing these kinetics are still not determined: In most cases, one can choose between several heuristics found in literature, or carry out experiments to determine them. In another model of the same process, one might neglect these phenomena and describe only the generalised fluxes in the reactor.

Furthermore, VeDa also includes a formalism for representing the steps the engineer executes while building a model.

Structural aspects [Souza1998]: VeDa divides the set of objects representing structural aspects of a process model into *devices* (such as a reactor or a distillation column) and *connections* (such as signal lines or valve connections). These classes can be further refined into more specific subclasses such as `heterogeneous-phase` or `directed-permeable-valve-connection`. It is not the case that each subclass is necessarily related to a specific function (such as mixing, reacting, etc.) the instances of this class have or a concrete gadget that can be found in chemicals plants. In most cases, subclasses are related to certain abstraction mechanisms that are employed in process modeling such as a heterogeneous, well-mixed, or homogeneous phases.

Objects representing structural aspects can be both atomic or aggregated. In order to appropriately support the top-down or bottom-up modeling, aggregated objects have to be described in such a way that, for example, the properties of the aggregated object can be deduced from the properties of its parts and from the way in which the interfaces of the parts match with the interfaces of the whole.

Behavioural aspects [Bogusch1998]: The behaviour of an object is, in a first place, described by the phenomena that are are associated to this object. These phenomena lead to the introduction of associated process quantities, which are related to each other by appropriate equations and which can be refined by constitutive equations. Hence the behavioural part of the VeDa class hierarchy contains classes for different types of phenomena, equations and variables.

Modeling steps [Lohmann1998; Krobb1997]: The development of a high-quality process model is a strongly creative procedure, hence it cannot be completely automated. However, it is possible to guide the engineer by, for example, proposing possible next steps which can be executed in the current state of the model. These steps as well as the description of modeling states are also formalised within VeDa. A *step* is characterised by

- a goal (such as the introduction of a new device),
- preconditions which have to hold for the step to be activated,
- postconditions which are guaranteed to hold after the execution of a step, and
- further properties.

An important property of this formalisation is that the syntax for the description of the pre- and postconditions was defined in such a way that (1) all realistic conditions can be expressed, and (2) the reasoning services such as checking whether a composite step is correctly decomposed into sub-steps, or checking whether one step can be substituted by another one are effectively computable.

Implementation of VeDa

In order to evaluate the VeDa approach, the modeling tool ModKit was developed and implemented at the Lehrstuhl für Prozeßtechnik of RWTH Aachen [Bogusch *et al.*1996]. It builds on the expert system shell G2 and is described in more detail in Section 5.1.

Chapter 3

Part-whole relations

As already mentioned, the process engineering application asks for a Description Logic with enough expressive power to represent the relevant properties of objects in the application domain, namely building blocks of process models. For the adequate representation of aggregated objects, the part-whole relation as well as those sub-part-whole relations relevant for the application have to be represented appropriately. This appropriateness can only be achieved if the properties of the part-whole relation and its sub-relations are known. In this chapter, we investigate in detail the properties of the part-whole relation and its subrelations, and present a general scheme for the classification of different kinds of part-whole relations. This scheme was partially developed while I was visiting A. Artale, N. Guarino, and E. Franconi at Ladseb-CNR in Padova, Italy.

The importance of part-whole relations in general led to the development of a philosophy based on the notion of parts and wholes, namely the *classical extensional mereology* as introduced in [Leśniewski1929] (for an introduction and exhaustive overview, see [Simons1987]). Classical extensional mereology is a formalism quite similar to set theory, with the difference that instead of being based on the membership relation, it is based on the part-whole relation. As for set theory, various logical formalisms are based on mereology, and different ways of axiomatising "the world" based on these formalisms were presented—differing, for example, in whether the world is supposed to be composed from atoms or not (where atoms are objects having no parts). Furthermore, mereological logics were also extended to deal with modalities, actions, and processes. Finally, [Simons1987] presents an exhaustive list of properties which can additionally hold between parts and wholes such as a part belonging exclusively to a whole.

Later, when the development in computer science gave rise to the field of knowledge representation, researchers in this field realised that part-whole relations have particular properties which have to be taken into account for the adequate representation of complex objects. This observation led to a great variety of investigations of these properties of part-whole relations [Winston *et al.*1987; Iris *et al.*1988; Gerstl & Pribbenow1993; Franconi1994; Padgham & Lambrix1994; Artale *et al.*1994; Pribbenow1995]—all motivated by the goal of developing a formalism for the adequate representation of complex objects.

A point of view supported by most of them is that there are different part-whole relations (such as component–aggregate and ingredient–object) with different properties, and that there is a general transitive part-whole relation containing all of these part-whole relations. Because of the particular properties of the different part-whole relations, the transitivity of the general part-whole relation, and the interaction between different specific part-whole relations, the part-whole relations cannot be represented adequately by simple binary relations. For example, consider a device d that has a part d' which, in turn, has a carcinogenic part z. A system answering "no" when asked whether d has a carcinogenic part can hardly be called adequate—but a system in which the part-whole relation is not modeled by a transitive relation can only answer with "no" to this question.

However, despite the fact that many authors believe in the relevance of different kinds of part-whole relations and underline the necessity to have a good understanding of the interaction between them, to our knowledge there is no "taxonomy" of part-whole relations. [Winston *et al.*1987] present different part-whole relations, but the authors only present examples to explain their meaning, and no exact definitions are given. From counterexamples—which strongly depend on the intuition of the reader—they infer that these different part-whole relations do not interact with each other. This is to say that from the fact that x is a part of y with respect to one sub-part-whole relation, and y is a part of z with respect to another sub-part-whole relation, one can only conclude that x is a part of z with respect to the general part-whole relation. It seems reasonable that this rather weak conclusion could be strengthened for some sub-part-whole relations.

In the following, a "taxonomy" of part-whole relations together with definitions of part-whole relations will be presented. From this framework, the way in which the part-whole relations interact follows by definition.

3.1 A well-defined taxonomy of part-whole relations

Given the need for a better understanding of the different part-whole relations, we developed a scheme for the classification of part-whole relations. The resulting taxonomy of part-whole relations is given in Figure 3.1. It shows classes of part-whole relations in a subclass/superclass hierarchy. At the top of this hierarchy, the general part-whole relation $\prec$ is found. It is even more general than mereological collections in that the principles imposed on mereological collections are stronger than the ones imposed on $\prec$ which is the union of all sub-part-whole relations. Hence $\prec$ strongly depends on these sub-relations, but the only property we want to impose on $\prec$ is that it is a strict partial order and thus transitive and irreflexive. These two properties are axiomatised by (A1) and (A2).

$$x \prec y \wedge y \prec z \Rightarrow x \prec z \tag{A1}$$

$$\neg x \prec x \tag{A2}$$

Mereological collections are presented in Section 3.1.1 whereas the other two sub-relations of $\prec$ are briefly presented in this section and in more detail in Section 3.1.2 and 3.1.3.

An *integral part-whole relation* is a subrelation of the general part-whole relation which involves some *integrity condition* on the parts. For x to be a part of y with respect to an integral part-whole relation, x has to be a part of y, and x has to satisfy the integrity condition associated with this relation. For example, for the *ingredient–mass* relation to hold between x and y, x has to be integral with respect to the substantial aspect. This means that y contains some material such as flour or oxygen and that x is exactly all of this material in y. Another example to mention is the *segment–entity* relation, which holds between z and d if z is a geometrically integral part of d. This means that z forms some geometric body within d. It is important to note that all these integrity conditions are imposed on the parts only, independent of the whole.

This independence is lacking for part-whole relations belonging to the second specialisation of the general part-whole relation. A *composed part-whole relation* is characterised by an additional relation which has to hold between a part and its whole. For example, the *component–aggregate* relation holds between x and y if x is a part of y, x is an integral object (with respect to some integrity condition) and x is functional for y, i.e., the functioning of y depends on x. This additional relation (such as being functional for in the previous example) is the reason why the composed part-whole relations are, in general, not transitive. If the additional relation is not transitive, then the composed part-whole relation cannot be transitive. For example, let a procedure be a component of (thus functional for) a small program, which is a component of a large program. If the procedure is never called by the large program because the large program uses the small one in a restricted way, then this procedure cannot be said to be a component of the large program.

All these relations can be specialised by refining the integrity conditions or the additional relations which have to hold between a part and its whole. As a consequence of the properties of integral and composed part-whole relations, integral part-whole relations are transitive and interact in a transitive way with other part-whole relations, whereas composed part-whole relations have, in general, none of these properties. In the sequel, these relations are defined in detail.

3.1.1 Mereological collections

From a mathematical point of view, a *mereology* is a partial order of a very particular kind. In this section, only the basic properties of this order are sketched. Most of the axioms given in the following that enforce these properties are taken from [Simons1987]. Let S be some *bag*[1] of individuals. Then classical mereology is built around a primitive part-whole relation $\ll$, which is axiomatised in the following using first-order logic with equality. The first two axioms enforce that $\ll$ is a strict partial order and (SA3) defines $\leqslant$ as the reflexive closure of $\ll$.

$$x \ll y \Rightarrow \neg y \ll x \tag{SA1}$$

$$x \ll y \wedge y \ll z \Rightarrow x \ll z \tag{SA2}$$

$$x \leqslant y \Leftrightarrow (x \ll y \vee x = y) \tag{SA3}$$

Still, this is axiomatisation is not sufficient. On the one hand, the *supplementation principle* says that if a whole has a strict part, then it must have another part different from the first one; see (SF1). However, (SF1) does not exclude ascending chains from being models. Axiom (SF2) enforces that each composed object has at least two parts which are not part of each other.

$$x \ll y \Rightarrow \exists z (z \ll y \wedge z \neq x) \tag{SF1}$$

$$x \ll y \Rightarrow \exists z (z \ll y \wedge \neg(z \leqslant x) \wedge \neg(x \leqslant z)) \tag{SF2}$$

[1] We use the notion "bag" instead of "set" to underline the fact that, in the following, notions like "subset" or "element" are not used in the set-theoretic sense.

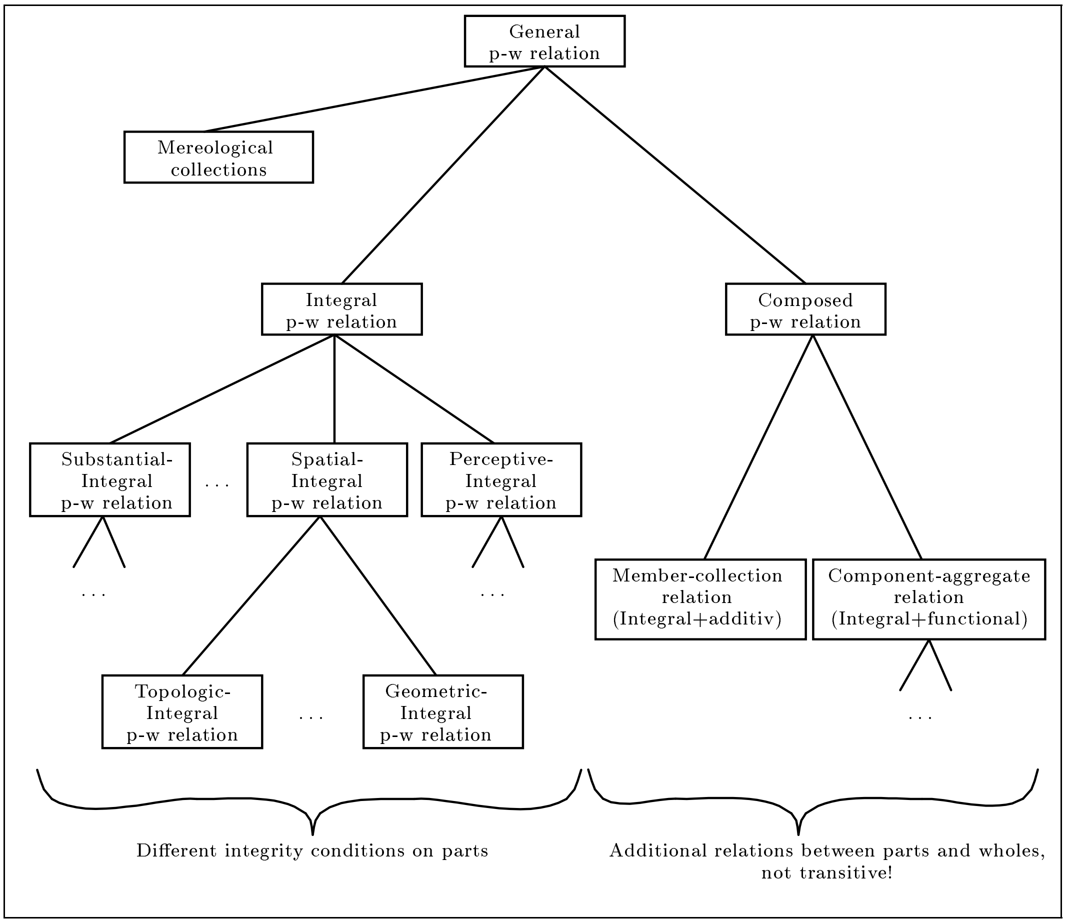

Figure 3.1: A taxonomy of part-whole relations.

Now, depending on the intuition one has of the relation $\ll$, these axioms are still not sufficient. Thus other axioms are added according to the ontological principles assumed. These principles determine the structures one accepts as models and those one wants to exclude from being models. For example, the *weak supplementation principle* says that if a whole has a part, then it has to have another part disjoint from the first one. This principle is axiomatised in (SA3), which, in contrast to (SF2), asks for an additional part z which has no part in common with x.

$$x \ll y \Rightarrow \exists z(z \ll y \land \neg(\exists w(w \ll z \land w \ll x))) \qquad \text{(SA3)}$$

Furthermore, one can restrict the models to those where, if two objects x, y have a part in common, then there exists another object which has x, y as proper parts. Another principle, *atomicity*, excludes structures with infinite ascending chains from being models. It assumes the existence of indivisible objects, so-called atoms, of which all other objects are composed. A last principle to be mentioned here is the *proper part principle*, which is similar to the extensionality principle of set theory. It assumes that no two composed objects have exactly the same set of strict parts. In this spirit, many other principles can be axiomatised, which lead to different mereologies and whose enumeration goes beyond the scope of this work; for details see [Simons1987]. However, the strong restrictions made on the order $\ll$ in all these mereologies have as a consequence that $\ll$ and its specialisations are specialisations of the general part-whole relation $\prec$.

3.1.2 Integral part-whole relations

An entity x is an *integral part* of an entity y with respect to the integrity condition Int if and only if x is a part of y (with respect to the general part-whole relation $\prec$) and if the integrity condition Int holds on x:

$$x \prec_{\mathsf{Int}} y \overset{\text{def.}}{\iff} x \prec y \wedge \mathsf{Int}(x).$$

The integrity condition Int is the most general integrity condition and simply imposes that x is integral with respect to some criteria, as for example its spatial extension ("the rectangle in the picture"), its substantial composition ("the flour in the cake"), the perception of an observer ("the nice part of the house"), etc. Whether an object x satisfies an integrity condition or not is independent of its context, hence from the transitivity of $\prec$ follows immediately that the integral part-whole relation is also transitive. Moreover, it interacts in a transitivity-like manner with $\prec$, that is

$$x \prec_{\mathsf{Int}} y \wedge y \prec z \Longrightarrow x \prec z \wedge \mathsf{Int}(x) \Longrightarrow x \prec_{\mathsf{Int}} z.$$

Refining the integrity conditions

The integral part-whole relation can be refined in a natural way by refining the integrity condition Int. For example, three ways of refinement are presented in Figure 3.1, namely according to whether a part x is integral with respect to

- its *spatial extension*, which means that x forms a connected body. This condition can be further refined by specifying whether a part is supposed to be integral with respect to topological conditions ("a convex segment of a cake"), to geometrical conditions ("a cubic segment of a cake"), whether it is allowed to contain holes or not, etc. The *segment-entity* relation discussed in [Gerstl & Pribbenow1993] belongs to this class of integral part-whole relations.

- its *substantial composition*. Again, we can refine this integrity condition to whether the part is integral with respect to its chemical composition ("the protein in a cake"), its physical state ("the ice cubes in a glass of water"), its molecular structure ("the diamonds in a heap of carbon"), etc.
- the *perception* of an observer. This condition can be further refined by specifying, for example, the sense organs involved in the perception ("the dark part of the room", "the loud part of the piece of music"), or whether a part is supposed to be integral with respect to emotional criteria ("the most beautiful moment in the holidays"), etc.

Let $\mathsf{SpecInt}$ be a refined integrity condition, i.e., $\mathsf{SpecInt}(x) \Rightarrow \mathsf{Int}(x)$. Then the corresponding refined integral part-whole relation $\prec_{\mathsf{SpezInt}}$ is defined similar to $\prec_{\mathsf{Int}}$, namely

$$x \prec_{\mathsf{SpezInt}} y \overset{\text{def.}}{\Longleftrightarrow} x \prec y \wedge \mathsf{SpecInt}(x).$$

Again, as for the general integral part-whole relation, refined integral part-whole relations interact in a transitivity-like manner with other part-whole relations, namely

$$x \prec_{\mathsf{SpezInt}} y \wedge y \prec z \Longrightarrow x \prec y \wedge \mathsf{SpecInt}(x) \wedge y \prec z \Longrightarrow x \prec_{\mathsf{SpezInt}} z.$$

Furthermore, if $\mathsf{SpecInt}'$ implies $\mathsf{SpecInt}$, then it is obvious that $x \prec_{\mathsf{SpecInt}'} y$ implies $x \prec_{\mathsf{SpezInt}} y$, hence the refinement of the integrity conditions yields a refinement of the integral part-whole relations in a straightforward way.

3.1.3 Composed part-whole relations

The third class of sub-relations of the general part-whole relation, *composed part-whole relations*, are characterised by the fact that they impose additional conditions which have to hold between a part and its whole. These additional conditions can no longer be viewed as integrity conditions on the part because they depend on both the part and its whole. One example for such an additional condition is functionality: A part engine can be both functional for a whole car (which works properly only if engine is present and intact) and non-functional for another whole garage (which has engine as its part, but works also without it). In general, these part-whole relations are of the form

$$x \prec_{\mathsf{Comp}} y \overset{\text{def.}}{\Longleftrightarrow} x \prec y \wedge x \;\mathsf{CompRel}\; y.$$

where CompRel is a binary relation characterising $\prec_{\mathsf{Comp}}$. For example, CompRel could be refined to Funct with the intended meaning that $x \;\mathsf{Funct}\; y$ holds if and only if x is functional for y. There are no restrictions on the relation CompRel, i.e., it has not to be transitive. As a consequence, composed part-whole relations are no longer transitive nor do they interact in a transitivity-like manner with other part-whole relations.

Another example for a composed part-whole relation is the so-called *member-collection* relation as described in [Winston *et al.*1987]. This relation holds, for example, between a tree and a forest, or between a grain of salt and a spoonful of salt. According to our understanding, an object x is a member of a collection y if and only if the following conditions are satisfied (see Figure 3.2 for a better intuition):

x is in $\{x_1, x_2, \dots\}$, a set of parts of y where each x_i satisfies a predicate Q (such as being a grain) which implies some kind of integrity condition. Furthermore, y is the sum of the objects x_i. This is to say that each part x' of y different from all x_i is either a part of some x_i or some x_i is a part of x' (see $(*)$ of the formal definition below). Furthermore, these x_i are supposed to be disjoint, that is they do not have any parts in common.

The formalisation of these conditions is rather complicated and involves a second-order variable Horizon which holds on the x_i that sum up to y. Furthermore, $\mathsf{Overlap}(z, w)$ is an abbreviation for $\exists v. v \prec z \wedge v \prec w$, and the member collection relation depends on the predicate Q.

$$\begin{array}{rcl}
x \prec^{Q}_{\mathsf{Member}} y & \overset{\text{def.}}{\Longleftrightarrow} & x \prec y \wedge \mathsf{Coll}_Q(x,y) \quad \text{where} \\
\mathsf{Coll}_Q(x,y) & \overset{\text{def.}}{\Longleftrightarrow} & \\
\exists \mathsf{Horizon}. & & \mathsf{Horizon}(x) \wedge \\
 & & \forall z.(\mathsf{Horizon}(z) \Rightarrow \\
 & & \quad Q(z) \wedge z \prec y \wedge \\
 & & \quad \forall w.((w \prec y \wedge (w \prec z \vee z \prec w)) \Rightarrow \neg\mathsf{Horizon}(w)) \wedge \\
 & & \quad \forall w.((w \prec y \wedge \mathsf{Horizon}(w)) \Rightarrow (\neg\mathsf{Overlap}(z,w) \vee z = w)) \wedge \\
 & & \forall z.(z \prec y \Rightarrow (\exists w.\mathsf{Horizon}(w) \wedge (z \prec w \vee w \prec z \vee w = z)))) \;\; (*)
\end{array}$$

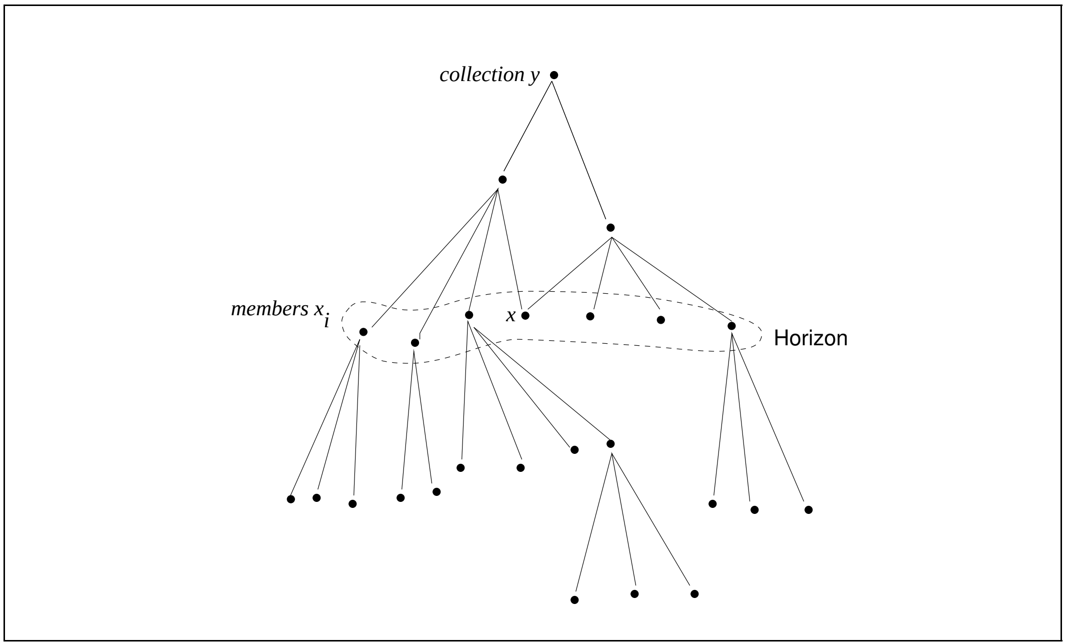

Figure 3.2: Example of a collection y and members x_i.

3.2 Comparison with other part-whole relations

In this section, the taxonomy of part-whole relations presented in [Winston *et al.*1987] is compared with the taxonomy presented in this section.

Component–Integral Object This relation is a composed part-whole relation with the additional condition that the part is functional for the whole. Formally, according to the definition in [Winston *et al.*1987], it can be written as $x \prec y \wedge x \ \mathsf{Functional}\ y$, where $x \ \mathsf{Functional}\ y$ holds only if x is integral with respect to some integrity conditions which is not specified in [Winston *et al.*1987]. Furthermore, the placement of the part within a whole might be relevant for the functionality—but this condition is not explained in more detail. The authors assume that this condition leads to a subrelation of the Component–Integral Object relation.

Member–Collection The Member–Collection relation defined in [Winston *et al.*1987] differs from the the one defined in Section 3.1.3 in that it does neither ask for any "similarity" between the members nor does it ensure disjointness of the members—instead, it only asks for spatial proximity or social connection. Hence the Member–Collection relation is also a composed part-whole relation in that it holds whenever a part (as well as all other parts) can be found in the same place or is "socially connected" to the whole. However, this does not really match the intuition given by the examples in [Winston *et al.*1987] such as tree-forest, ship-fleet or juror-jury: These examples fit better to the member-collection relation defined in Section 3.1.3 because the forest (resp. jury, fleet) is the sum of trees (resp. jurors, ships), and the forest (resp. jury, fleet) can be decomposed into trees (resp. jurors, ships) which are disjoint from each other and similar to each other.

Portion–Mass The example given in [Winston *et al.*1987] for this relation is "a slice of pie" and it is characterised by the fact that the part (called piece here) is not functional for the whole, whereas it has to be similar to other pieces as well as to the whole in being pie. The authors say that, if the Portion–Mass relation holds between two objects x, y, then one can say "x is some of y" as in "Could you please pass me some of the pie?", where it would be incorrect to pass only the crust of the pie. Since this relation strongly depends on the similarity between the portion and the whole, it is parametrised by a predicate Q which, in the example, is being pie. The Portion–Mass relation can be defined as follows and thus belongs to the composed part-whole relations.

$$\begin{aligned} x \prec^{Q}_{\mathsf{Portion}} y \iff & \; x \prec y \wedge \mathsf{Piece}_Q(x,y) \quad \text{where} \\ \mathsf{Piece}_Q(x,y) \iff & \; Q(x) \wedge Q(y) \wedge \exists z.(z \prec y \wedge Q(z) \wedge \neg \mathsf{Overlap}(x,z) \wedge \\ & \quad \forall w.(w \prec y \Rightarrow (\mathsf{Overlap}(x,w) \vee \mathsf{Overlap}(z,w)))) \end{aligned}$$

Stuff–Object According to [Winston *et al.*1987], if this relation holds between x and y, one can say that "x is made of y", and y cannot be removed without altering y's identity. Unfortunately, the authors do not explain what a thing's identity is. Examples given are alcohol-martini, steel-bike, and hydrogen-water. In the taxonomy presented in this chapter, Stuff–Object is clearly a substantial-integral part-whole relation.

Feature–Activity The only difference between component–integral object and feature–activity is that both the part and the whole of the latter have to be actions. It appears that this difference were made only because the inverse of the former translates in English to "has" whereas this translation does not work for the latter.

Place–Area This relation between special places and areas is said to be similar to the portion–mass relation besides the fact that places cannot be separated from the area they occupy in the sense a slice of pie can be taken away from a pie. The authors distinguish between Place–Area and the topological inclusion relation. The first one also demands that the place is part of the area it can be found in whereas for being topologically included by something it suffices to be surrounded by it.

Consequences for knowledge representation issues

As we have seen in this section, there is a great variety of different part-whole relations with different properties, and some of them are rather complicated, such as the member-collection relation or the component-aggregate relation. The first one is complicated since it describes a whole as the sum of disjoint parts, all of them satisfying some additional integrity condition. The second one is, at a first glance, less complicated, but the description or axiomatisation of "functionality" is very complex: It requires the definition of "correctly functioning". The expressive power of decidable formalisms such as Description Logics is surely too weak to define the member-collection or the component-aggregate relation.

The influences of the ontological principles on the required expressive power are only sketched for two examples, namely the proper part and the atomicity principle: If a Description Logic has the tree model property (see Section 4.5.3 for a definition of this property), then a concept C is satisfiable iff C is satisfiable in a model obeying the proper part principle: Each satisfiable concept has a tree model which obviously obeys the proper part principle. Other inference problems (such as subsumption) are either likely to be reducible to satisfiability—in this case they are also invariant under the assumption of the proper part principle—or the corresponding inference algorithms can be easily extended to obey the proper part principle (this is the case for ABox consistency). Similarly, the atomicity principle comes for free in Description Logics having the finite tree model property: Satisfiable concepts of such a Description Logic are always satisfiable in a model that has no infinite ascending chain. If a Description Logic does not have the finite tree model property, then one can enforce infinite ascending chains, and special techniques have to be applied for testing satisfiability in a model obeying the atomicity principle. Such a technique is described in [Calvanese1996].

The investigations of the requirements made by all these principles for the expressive power of Description Logics goes beyond the scope of this work. We concentrated on different possibilities to extend Description Logics by transitive relations, and to relate transitive relations to other, possibly non-transitive relations. The results of these investigations together with the consequences of these results for the representation of part-whole relations can be found in Chapter 7.

Chapter 4

Description Logics

In this section, the basic Description Logic $\mathcal{ALC}$ is formally introduced. Starting with a short history of Description Logics, syntax and semantics of $\mathcal{ALC}$ are introduced as well as a formal definition of the corresponding inference problems. For a better understanding of the inference algorithms presented later, the reasoning techniques used within this work are first given for $\mathcal{ALC}$. Then it is explained how these reasoning techniques can be used within a knowledge representation system based on Description Logics. The main ideas of knowledge representation systems based on Description Logics are pointed out, including the description of the main reasoning services of such systems. Next, the extensions of $\mathcal{ALC}$ by (1) number restrictions and by (2) the transitive closure of roles are introduced. Finally, the expressive power and some model theoretic properties of these basic languages are discussed.

4.1 A brief history of Description Logics

Since the late eighties, Description Logics play an important rôle in the field of knowledge representation. The origin of Description Logics can be said to lie in the necessity to formalise the semantics of *semantic networks* [Sowa 1987]: This is a vivid, graphical representation formalism where concepts and individuals are related to each other via labelled edges. For example, the semantic network given in Figure 4.1 states that elephants are a specialisation of mammals, and that Clyde is an elephant. Despite the vividness of semantic networks, their lack in formal semantics became irritating especially when trying to understand answers given by knowledge representation systems based on semantic networks [Brachman1983]. To overcome this problem,

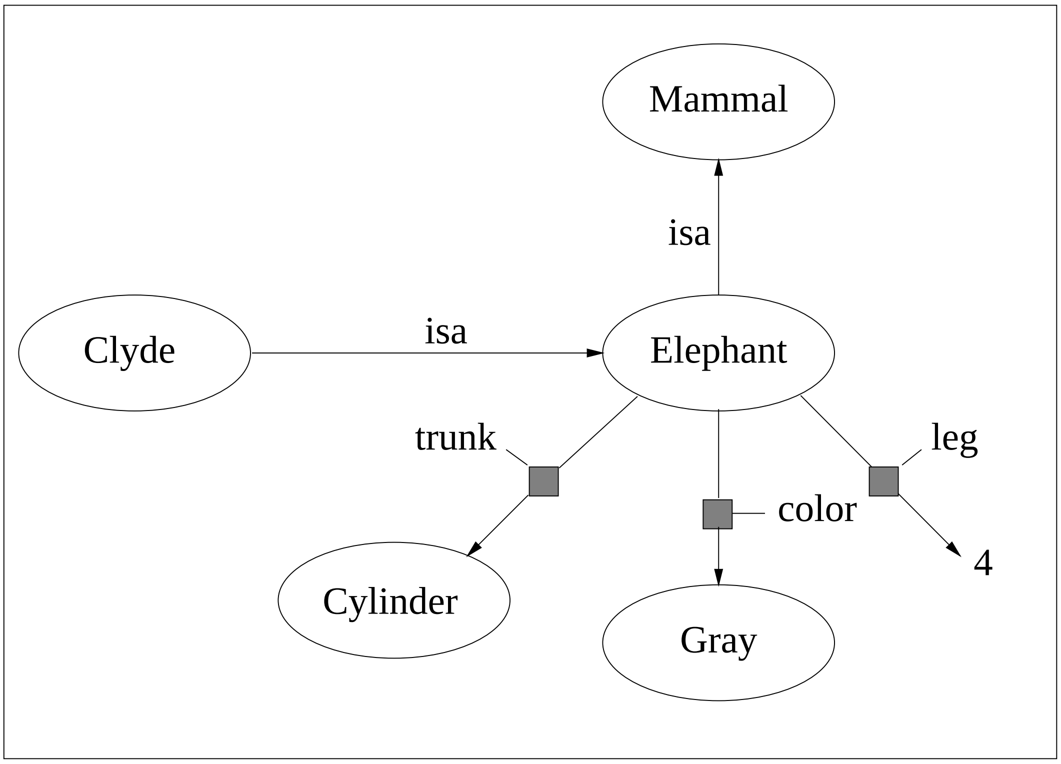

Figure 4.1: A semantic network.

a new graphical formalism, *structured inheritance networks*, was introduced and implemented [Brachman *et al.*1991; Kaczmarek *et al.*1986]. Furthermore, structured inheritance networks were provided with a well-defined semantics which fixed the precise meaning of its graphical constructs—which led to the definition of the first *Description Logics* [Brachman & Schmolze1985].

Unfortunately, it turned out that the main inference problems of these first Description Logics were undecidable [Patel-Schneider1989; Schmidt-Schauss 1989]. This insight and the fact that sound and complete inference algorithms seem to be crucial for the tasks knowledge representation systems based on Description Logics are applied to led to the restriction of the expressive power of subsequent Description Logics and to numerous investigations concerning their computational complexity [Hollunder *et al.*1990; Baader1991; Baader & Hanschke1991; Donini *et al.*1991a; 1991b; Hollunder & Baader1991; Schmidt-Schauß & Smolka1991; Hanschke1992; Baader *et al.*1993; Baader & Hanschke 1993; Baader & Sattler1996a; Sattler1996; Baader & Sattler1996b; De Giacomo *et al.*1996]

In 1991, the close relationship between Modal Logics and Description Logics was discovered and published [Schild1991; 1994; De Giacomo & Lenzerini 1994a], which led to new complexity results especially in the field of Description Logics [De Giacomo & Lenzerini1994b; De Giacomo1995].

In a nutshell, these investigations revealed that a tractable Description Logic is of rather low expressive power, and that extensions which add some more expressiveness lead at least to PSPACE-completeness. However, all these results only concern worst-case complexity. Several knowledge representation systems based on intractable Description Logics are used in applications [Rychtyckyj 1996; Kirk *et al.*1995], and tests were undertaken which have shown that Description Logics with a high worst-case complexity behave quite well on real-world or random knowledge bases [Bresciani *et al.*1995; Baader *et al.*1994; Horrocks1997].

The relationship between Description Logics and other class-based representation formalisms such as object-oriented databases, entity relationship diagrams, frame based systems has been investigated since 1991. On the one hand, Description Logics can be viewed as a unified framework for these formalism, with the profit that the precise semantics of Description Logics is inherited to these formalisms and that reasoning techniques developed for Description Logics can be used for reasoning in these formalisms [Bergamaschi & Sartori1992; Buchheit *et al.*1994; Calvanese *et al.*1994; Goñi *et al.*1996; Lenzerini & Schaerf 1991; Levy *et al.*1996; Beeri *et al.*1997]. On the other hand, several extensions of Description Logics were motivated by the necessity to capture some aspects of the expressiveness of other formalisms [Calvanese & Lenzerini1994; Calvanese *et al.*1995; De Giacomo & Lenzerini1995].

4.2 The basic Description Logic $\mathcal{ALC}$

The Description Logic underlying all investigations in this work is $\mathcal{ALC}$, a well-known Description Logic introduced and investigated in [Schmidt-Schauß & Smolka1991]. Further investigations concerning extensions and restrictions of $\mathcal{ALC}$ can be found, for example, in [Hollunder *et al.*1990; Donini *et al.*1991a; 1995]. In $\mathcal{ALC}$, concepts can be built using

- propositional operators, i.e., *and* ($\sqcap$), *or* ($\sqcup$), and *not* ($\neg$),
- value restrictions on those individuals associated to an individual via a certain role. This includes *existential* restrictions as in ($\exists$ `has_child.Girl`) as well as *universal* restrictions such as ($\forall$ `has_child.Human`).

Definition 4.1 Let N_C be a set of *concept names* and let N_R be a set of *role names*. The set of $\mathcal{ALC}$-*concepts* is the smallest set such that

1. every concept name is a concept and
2. if C and D are concepts and R is a role name, then $(C \sqcap D)$, $(C \sqcup D)$, $(\neg C)$, $(\forall R.C)$, $(\exists R.C)$ are concepts.

In order to fix the exact meaning of these concepts, their semantics is given in a model-theoretic way.

Definition 4.2 An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a set $\Delta^{\mathcal{I}}$, called the *domain* of $\mathcal{I}$, and a function $\cdot^{\mathcal{I}}$ which maps every concept to a subset of $\Delta^{\mathcal{I}}$ and every role to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ such that

$$
\begin{aligned}
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}}, \\
(C \sqcup D)^{\mathcal{I}} &= C^{\mathcal{I}} \cup D^{\mathcal{I}}, \\
\neg C^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}, \\
(\exists R.C)^{\mathcal{I}} &= \{d \in \Delta^{\mathcal{I}} \mid \text{There exists an } e \in \Delta^{\mathcal{I}} \text{ with } (d,e) \in R^{\mathcal{I}} \text{ and } e \in C^{\mathcal{I}}\}, \\
(\forall R.C)^{\mathcal{I}} &= \{d \in \Delta^{\mathcal{I}} \mid \text{For all } e \in \Delta^{\mathcal{I}}, \text{ if } (d,e) \in R^{\mathcal{I}}, \text{ then } e \in C^{\mathcal{I}}\}.
\end{aligned}
$$

A concept C is called *satisfiable* iff there is some interpretation $\mathcal{I}$ such that $C^{\mathcal{I}} \neq \emptyset$. Such an interpretation is called a *model* of C. A concept D *subsumes* a concept C (written $C \sqsubseteq D$) iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds for each interpretation $\mathcal{I}$. Two concepts are said to be equivalent (written $C \equiv D$) if they mutually subsume each other.

For an interpretation $\mathcal{I}$, an individual $x \in \Delta^{\mathcal{I}}$ is called an *instance* of a concept C iff $x \in C^{\mathcal{I}}$. Finally, x is said to be an R-successor of an individual $w \in \Delta^{\mathcal{I}}$ iff $(w, x) \in R^{\mathcal{I}}$.

Additional propositional operators, such as implication, will be used as abbreviations: for example, $A \Rightarrow B$ stands for $\neg A \sqcup B$. In complex concepts (such as those used in the reductions later), these abbreviations increase the readability. Furthermore, $\top$ is used as an abbreviation for the universal concept, i.e., $\top = A \sqcup \neg A$, and $\bot$ is used as an abbreviation for the empty concept, i.e., $\bot = A \sqcap \neg A$.

Remark 4.3 An alternative for the presentation of the semantics of $\mathcal{ALC}$ is to translate concepts into first-order formulae with one free variable. This translation can be defined in such a way that the resulting formulae involve only 2 variables x, y. The translation is given by two translation mappings t_x, t_y from $\mathcal{ALC}$-concepts into the set of first-order formulae over two variables. For each concept name A, a unary predicate ϕ_A is introduced, and for each role name R, a binary relation ρ_R is introduced. For C, D possibly complex concepts, the translation is defined as follows:

$$\begin{array}{rclrcl}
t_x(A) & = & \phi_A(x), & t_y(A) & = & \phi_A(y), \\
t_x(C \sqcap D) & = & t_x(C) \wedge t_x(D), & t_y(C \sqcap D) & = & t_y(C) \wedge t_y(D), \\
t_x(C \sqcup D) & = & t_x(C) \vee t_x(D), & t_y(C \sqcup D) & = & t_y(C) \vee t_y(D), \\
t_x(\exists R.C) & = & \exists y.\rho_R(x,y) \wedge t_y(C), & t_y(\exists R.C) & = & \exists x.\rho_R(y,x) \wedge t_x(C), \\
t_x(\forall R.C) & = & \forall y.\rho_R(x,y) \Rightarrow t_y(C), & t_y(\forall R.C) & = & \forall x.\rho_R(y,x) \Rightarrow t_x(C).
\end{array}$$

Obviously, a concept C is satisfiable iff its translation $t_x(C)$ is satisfiable, and a concept C is subsumed by a concept D iff $t_x(C \Rightarrow D)$ is valid.

Remark 4.4 If a Description Logic allows for negation and conjunction of concepts, such as $\mathcal{ALC}$, subsumption and (un)satisfiability can be reduced to each other:

- $C \sqsubseteq D$ iff $C \sqcap \neg D$ is unsatisfiable,
- C is unsatisfiable iff $C \sqsubseteq A \sqcap \neg A$ (for some concept name A).

Since most Description Logics considered here are in fact propositionally closed, this connection between satisfiability and subsumption will be heavily exploited: We may restrict our attention to one of these problems, both in the decidability and in the undecidability proofs.

4.3 Knowledge representation based on Description Logics

Until now, Description Logics were introduced without mentioning how these Logics can be used for knowledge representation. In this section, the main ideas concerning knowledge representation based on Description Logics are introduced, as well as an example (for more details, see [Nebel1990]). Roughly spoken, knowledge representation systems based on Description Logics (DL systems for short) consist of 3 parts:

- a reasoner, often based on a tableau algorithm,

- knowledge bases in which the user models the explicit knowledge of an application domain, and
- interfaces to the functionalities of the system.

In most DL systems, there are two different kinds of knowledge bases:

(1) The *terminological* knowledge of an application domain is stored in the so-called TBox. To this purpose, the TBox contains a set of concept definitions, which are, in their simple version, of the form $A := C$ for a concept name $A \in N_C$ and a (possibly complex) concept C. Concepts occurring on the left-hand side of a concept definition are called *defined* concept, and it depends on the underlying Description Logic which constructors can be used to build the concepts on the right-hand side.

(2) In addition to the terminological knowledge described in a TBox, concrete situations can be described in the so-called ABox, possibly using the concepts defined in the TBox. In order to describe concrete situations, *individual* names $a, b, \ldots$ are used to refer explicitly to elements of an interpretation domain. Then ABox-statements can express that an individual a is an instance of a concept C (written $a : C$), or that two individuals a, b are related via a role R (written aRb). The similarity between ABox statements and constraints of the completion algorithm is not incidental. Constraint systems can be viewed as special cases of ABoxes, where the specialisation is due to the fact that constraint systems for $\mathcal{ALC}$ and several extension do not contain cycles, whereas ABoxes are not required to be acyclic.[1]

It is straightforward to extend the notion of a *model* to a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$: An interpretation $\mathcal{I}$ associates, additionally, each individual a occurring in $\mathcal{A}$ to an element $\pi(a) \in \Delta^{\mathcal{I}}$. Then $\mathcal{I}, \pi$ satisfy $\mathcal{A}$ and $\mathcal{T}$ iff they satisfy each of their statements. This is to say that $\mathcal{I}, \pi$ have the following properties:

$$\begin{array}{lllcl} \text{for all } A := C & \text{in } \mathcal{T}: & A^{\mathcal{I}} & = & C^{\mathcal{I}}, \\ \text{for all } a : C & \text{in } \mathcal{A}: & \pi(a) & \in & C^{\mathcal{I}}, \\ \text{for all } aRb & \text{in } \mathcal{A}: & (\pi(a), \pi(b)) & \in & R^{\mathcal{I}}. \end{array}$$

Such an interpretation is then called a model of $\mathcal{A}$ and $\mathcal{T}$. Furthermore, a concept C is said to be *$\mathcal{T}$-subsumed* by a concept D (written $C \sqsubseteq_{\mathcal{T}} D$) iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds for all models $\mathcal{I}$ of $\mathcal{T}$. An individual a is said to be an *instance* of a concept C with respect to $\mathcal{A}$ and $\mathcal{T}$ iff $a \in C^{\mathcal{I}}$ for all models of $\mathcal{A}$ and $\mathcal{T}$. Finally, an ABox $\mathcal{A}$ is called *consistent* with a TBox $\mathcal{T}$ iff there exists a model for both, $\mathcal{A}$ and $\mathcal{T}$.

[1] A cyclic ABox contains statements $a_0 R_1 a_1, a_1 R_2 a_2, \ldots, a_{n+1} R_n a_0$.

DL systems vary in whether they allow for *cyclic* concept definitions in a TBox or not: A concept A is said to "use" a concept C if C occurs on the right hand side of A's definition. If the transitive closure of "uses" contains a concept that "uses" itself, this TBox is called cyclic. For acyclic TBoxes, reasoning with respect to a TBox $\mathcal{T}$ can be reduced to reasoning with respect to the empty TBox by *unfolding* (see [Nebel1990]). Unfolding means substituting all concepts A_i which are defined in the TBox by $A_i := C_i$ by their definition C_i. Applying these substitutions to the concepts occurring in an ABox, consistency with respect to a TBox can thus be reduced to consistency with respect to the empty TBox. Similar, subsumption and satisfiability of concepts, and the problem whether an individual is an instance of a given concept with respect to a TBox can be reduced to the corresponding problems with respect to the empty TBox.

In general, DL systems employ the *unique-name-assumption*, which means that a model has to interpret different individual names of $\mathcal{A}$ as different elements of the interpretation domain. Furthermore, the *open-world-assumption* allows that statements which are not implied by an ABox and a TBox may hold or may not hold. This will become more clear in the next example.

Example 4.5 The TBox given in Figure 4.2 contains some basic concept definitions of the process engineering application. There, the behavioural aspects of a device or a connection are separated from their structural aspects by storing the first one in its implementation and the second aspect in its interfaces:

This example is incomplete in that concepts such as `device_implementation` or `atomic` are not defined, hence there are models of this knowledge base which are not conform to the engineer's intuition. It is easy to see that `Device` subsumes both `Atomic_device` and `Comp_device`. A closer investigation reveals that `Atomic_device` and `Comp_device` are disjoint, which means that an ABox containing `a:Atomic_device` and `a:Comp_device` is inconsistent with this TBox. This is due to the fact that a device must be implemented by an implementation, which must have a description of its behaviour, and that this behaviour description is atomic for an instance of `Atomic_device` and not atomic for an instance of `Comp_device`.

The ABox in Figure 4.3 describes some objects of a process model. Since the concepts on the right hand side of a ":" are not restricted to defined concepts, they can be complex concepts as in the first statement.

With respect to the above given knowledge base, the question "Is `valve7` an instance of `Comp_device`?" must be answered with "no": One can construct a model of this ABox and TBox, where `valve7` has a third part which is neither

$$
\begin{aligned}
\texttt{Device} &:= (\exists \texttt{implemented\_by} \,.\, \texttt{Device\_implementation}) \sqcap \\
&\quad (\exists \texttt{interfaces} \,.\, \texttt{Device\_interface}) \\
\texttt{Connection} &:= (\exists \texttt{implemented\_by} \,.\, \texttt{Connection\_implementation}) \sqcap \\
&\quad (\exists \texttt{interfaces} \,.\, \texttt{Connection\_interfaces}) \sqcap \neg \texttt{Device} \\
\texttt{Comp\_device} &:= \texttt{Device} \sqcap (\forall \texttt{has\_part} \,.\, (\texttt{Device} \sqcup \texttt{Connection})) \sqcap \\
&\quad (\forall \texttt{implemented\_by} \,.\, \texttt{Comp\_device\_implementation}) \\
\texttt{Atomic\_device} &:= \texttt{Device} \sqcap \neg(\exists \texttt{has\_part} \,.\, \texttt{Anything}) \sqcap \\
&\quad (\forall \texttt{implemented\_by} \,.\, \texttt{Atomic\_device\_implementation}) \\
\texttt{Device\_implementation} & \\
&:= \texttt{Implementation} \sqcap (\exists \texttt{behaviour} \,.\, \texttt{Behav\_descr}) \\
\texttt{Atomic\_device\_implementation} & \\
&:= \texttt{Device\_implementation} \sqcap (\forall \texttt{behaviour} \,.\, \texttt{Atomic}) \\
\texttt{Comp\_device\_implementation} & \\
&:= \texttt{Device\_implementation} \sqcap (\forall \texttt{behaviour} \,.\, \neg \texttt{Atomic})
\end{aligned}
$$

Figure 4.2: Example TBox.

$$
\begin{aligned}
\texttt{valve7} &: \texttt{Device} \sqcap \\
&\quad (\exists \texttt{implemented\_by} \,.\, \texttt{Valve\_implementation}) \\
\texttt{float01} &: \texttt{Device} \\
\texttt{chamber13} &: \texttt{Device} \\
\texttt{valve\_implementat03} &: \texttt{Comp\_device\_implementat} \\
&\texttt{valve7 has\_part float01} \\
&\texttt{valve7 has\_part chamber13} \\
&\texttt{valve7 implemented\_by valve\_implementation03}
\end{aligned}
$$

Figure 4.3: Example ABox.

a connection nor a device. This is due to the above mentioned open-world-assumption: Everything implied by the statements in a knowledge base has to hold in each of its models, but propositions which are neither implied by these statements nor contradicting them might hold or might not hold in its models.

The basic system services and inference problems of a DL system are the following:

Classification: Computes the ordering of the concepts defined in a TBox with respect to the subsumption relation.

Subsumption: Tests whether a given concept subsumes another given concept, possibly with respect to a TBox.

Satisfiability: Checks whether a given concept can ever be interpreted as a non-empty set, possibly with respect to a TBox.

Consistency: Tests whether a given ABox is consistent with respect to a given TBox.

Retrieval: Finds all those individuals of a given ABox that are instances of a given concept.

Most-specific-concepts: Given an individual, it computes the most specific classes this individual is an instance of.

In the following, this work will focus on satisfiability problems for the following reasons. Given that the Description Logics to be investigated are mostly propositionally closed,

- undecidability of satisfiability of concepts implies that none of the above mentioned system services are computable.
- subsumption can be reduced to satisfiability, hence algorithms deciding satisfiability can be used for deciding subsumption; see Remark 4.4.

Furthermore, this process engineering application does not ask for cyclic definitions if transitive roles are present. Hence subsumption with respect to a TBox can be reduced to subsumption, which can be reduced to satisfiability for Description Logic that are propositionally closed.

Given that the TBox is acyclic, the only inference problems that can not be reduced to concept satisfiability are the ones involving ABoxes, namely consistency, retrieval and most-specific-concepts. Ignoring these problems was, on the one hand, the only way to investigate several extensions relevant for the process engineering application—including ABox reasoning to the set of problems to be investigated would have gone beyond the scope of this work. On the other hand, tableau-based algorithms deciding concept satisfiability can be used as a basis for the development of algorithms for ABox inference problems.

For many Description Logics, tableau-based satisfiability algorithms can be easily modified such that they are able to decide ABox consistency. This is true for several extensions of $\mathcal{ALC}$, namely for those by simple and qualifying number restrictions [Hollunder1994], admissible concrete domains [Baader & Hanschke1991], admissible concrete domains and generalised existential and universal value restrictions [Hanschke1992], by inverse and boolean operators on roles [De Giacomo1995], and many more.

4.4 A tableau-based algorithm for $\mathcal{ALC}$

In general, to show decidability of satisfiability of concepts of a Description Logic $\mathcal{D}$, one has two possibilities:

- One gives a translation from $\mathcal{D}$-concepts to $\mathcal{D}'$-concepts, where $\mathcal{D}'$ is a Description Logic whose satisfiability problem is known to be decidable. This translation has to be invariant for satisfiability, i.e., a $\mathcal{D}$-concept C is satisfiable if and only if its translation C' is a satisfiable $\mathcal{D}'$-concept. This technique is applied, for example, in [De Giacomo1995].

- One presents an algorithm and shows that it decides satisfiability of $\mathcal{D}$-concepts. That is, started with some $\mathcal{D}$-concept C, it always terminates, and it answers with "C is satisfiable" if and only if C is satisfiable. Such an algorithm is called *sound* and *complete*, where soundness refers to the fact that all concepts said to be satisfiable are indeed satisfiable, and completeness refers to the fact that all satisfiable concepts are found to be satisfiable.

Throughout this work, the second possibility has always been chosen, and several decision algorithms are presented. All of them are tableau-based, and the techniques used to prove their termination, soundness, and completeness are quite similar. In order to understand these techniques better and to make their presentation easier, they are demonstrated first for $\mathcal{ALC}$.

Please note that these algorithms are not thought to be implemented directly, but to serve as an implementation base. Several optimisation techniques have already been developed and tested, and their impact on the performance of these algorithms has been evaluated. In a nutshell, it turned out that these techniques have an enormous influence on the efficiency of these algorithms. Especially techniques for the efficient treatment of the propositional part of Description Logics [Giunchiglia & Sebastiani1996] and several techniques for early clash detection [Baader *et al.*1994; Bresciani *et al.*1995; Horrocks1997] turned out to be extremely powerful. The algorithms presented here are designed in such a way that interesting properties such as soundness, completeness, and termination can be proved in a rather elegant way.

The tableau-based algorithm presented in the following decides satisfiability of $\mathcal{ALC}$-concepts. Given an $\mathcal{ALC}$-concept C_0, it tries to build a model for C_0. It breaks C_0 down to its subconcepts and tries to satisfy C_0 by satisfying, step by step, the constraints induced by subconcepts of C_0. If all these attempts fail, C_0 is unsatisfiable, otherwise, one can easily build a model for C_0 from the structures generated while breaking down C_0.

For simplicity, all concepts are supposed to be in *negation normal form* (NNF for short). This means that negation is applied to concept names only. An $\mathcal{ALC}$-concept can be transformed into an equivalent one in NNF by using the following equivalences to push negation into concepts.

$$\begin{aligned}
\neg(C \sqcup D) &\equiv \neg C \sqcap \neg D \\
\neg(C \sqcap D) &\equiv \neg C \sqcup \neg D \\
\neg(\exists R.C) &\equiv (\forall R.\neg C) \\
\neg(\forall R.C) &\equiv (\exists R.\neg C)
\end{aligned}$$

The basic data structure our algorithm works on are *constraints*. For $\mathcal{ALC}$, we use two different kinds of constraints: For example, the constraint x `connected-to` y ensures that the two individuals x and y stand for are related via the role `connected-to`, and the constraint x : `Device` ensures that the individual x stands for is an instance of the concept `Device`.

Definition 4.6 Let $\tau = \{x, y, z, \dots\}$ be a countably infinite set of individual variables. A *constraint* is either of the form

xRy, where R is a role name in N_R and $x, y \in \tau$,

$x : D$ for some $\mathcal{ALC}$-concept D and some $x \in \tau$.

A *constraint system* is a set of constraints. For a constraint system S, let $\tau_S \subseteq \tau$ denote the individual variables occurring in S.

An interpretation $\mathcal{I}$ is a *model of a constraint system* S iff there is a mapping $\pi : \tau_S \to \Delta^{\mathcal{I}}$ such that $\mathcal{I}, \pi$ satisfy each constraint in S, i.e.,

$$\begin{aligned}
(\pi(x), \pi(y)) \in R^{\mathcal{I}} &\quad \text{for all } xRy \in S, \\
\pi(x) \in D^{\mathcal{I}} &\quad \text{for all } x : D \in S.
\end{aligned}$$

1. Conjunction: If $x:(C_1 \sqcap C_2) \in S$ and $x:C_1 \notin S$ or $x:C_2 \notin S$, then
$S \rightarrow \quad S \cup \{x:C_1, x:C_2\}$

2. Disjunction: If $x:(C_1 \sqcup C_2) \in S$ and $x:C_1 \notin S$ and $x:C_2 \notin S$, then
$S \rightarrow \quad S_1 = S \cup \{x:C_1\}$
$S \rightarrow \quad S_2 = S \cup \{x:C_2\}$

3. Value restriction: If $x:(\forall R.C) \in S$, y is an R-successor of x in S and $y:C \notin S$, then
$S \rightarrow \quad S \cup \{y:C\}$

4. Existential restriction: If $x:(\exists R.C) \in S$, and there is no R-successor y of x in S with $y : C \in S$, then
$S \rightarrow \quad S \cup \{xRz, z : C\}$ for a new variable $z \in \tau \setminus \tau_S$.

Figure 4.4: The completion rules for $\mathcal{ALC}$.

For a constraint system S, individual variables x, y, and role names R_i, we say that y is an $R_1 \circ \ldots \circ R_m$-*successor* of x in S iff there are $y_0, \ldots, y_m \in \tau$ such that $x = y_0, y = y_m$, and $\{y_i R_{i+1} y_{i+1} \mid 0 \leq i \leq m-1\} \subseteq S$. S contains a *clash* iff $\{x:A, x:\neg A\} \subseteq S$ for some concept name A and some variable $x \in \tau_S$. A constraint system S is called *complete* iff none of the completion rules given in Figure 4.4 can be applied to S.

Figure 4.4 introduces the *completion rules* that are used to test $\mathcal{ALC}$-concepts for satisfiability. The *completion algorithm* works on a tree where each node is labelled with a constraint system. It starts with the tree consisting of a single leaf, the root, labelled with $S = \{x_0 : C_0\}$, where C_0 is the $\mathcal{ALC}$-concept in NNF to be tested for satisfiability. A rule can only be applied to a leaf labelled with a clash-free constraint system. Applying a rule $S \rightarrow S_i$, for $1 \leq i \leq n$, to such a leaf leads to the creation of n new successors of this node, each labelled with one of the constraint systems S_i. The algorithm terminates if none of the rules can be applied to any of the leaves. In this situation, it answers with "C_0 is satisfiable" iff one of the leaves is labelled with a clash-free constraint system.

Example 4.7 If this algorithm tests

$$C_0 := A \sqcap (\forall R.(C \sqcup D)) \sqcap (\exists R.\neg C)$$

for satisfiability, it starts with the constraint system $\{x : C_0\}$, and after two applications of Rule 1, we have

$$S' = \{x : C_0,\ x : (\forall R.(C \sqcup D)) \sqcap (\exists R.\neg C),\ x : A, x : (\forall R.(C \sqcup D)),\ x : (\exists R.\neg C)\}.$$

Next, only Rule 4 can be applied, which adds xRy and $y : \neg C$ to S. Then Rule 3 can be applied, which adds $y : (C \sqcup D)$ and yields S''. Finally, Rule 2 yields

$$\begin{array}{rcl} S_1 & = & S'' \cup \{y : C\}, \\ S_2 & = & S'' \cup \{y : D\}. \end{array}$$

The constraint system S_1 contains a clash because $\{y : C, y : \neg C\} \subseteq S_1$. In contrast, S_2 does not contain a clash, nor can any rule be applied to it. Hence it is a complete, clash-free constraint system, and C_0 is satisfiable. Furthermore, S_2 can be used to build a model $\mathcal{I}$ of C_0. The interpretation domain $\Delta^{\mathcal{I}}$ is simply the set of variables occurring in S_1, and concepts as well as roles are interpreted according to the constraints in S_1, i.e.,

$$\begin{array}{rclcl} \Delta^{\mathcal{I}} & = & \tau_{S_1} & = & \{x, y\}, \\ A^{\mathcal{I}} & = & \{x \in \Delta^{\mathcal{I}} \mid x : A \in S_1\} & = & \{x\}, \\ D^{\mathcal{I}} & = & \{x \in \Delta^{\mathcal{I}} \mid x : D \in S_1\} & = & \{y\}, \\ C^{\mathcal{I}} & = & \{x \in \Delta^{\mathcal{I}} \mid x : C \in S_1\} & = & \emptyset, \\ R^{\mathcal{I}} & = & \{(x, y) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid xRy \in S_1\} & = & \{(x, y)\}. \end{array}$$

In order to show that this completion algorithm really yields a decision procedure for $\mathcal{ALC}$, it suffices to proof the following lemma.

Lemma 4.8 Let C_0 be an $\mathcal{ALC}$-concept in NNF, and let S be a constraint system obtained by applying the completion rules to $\{x_0 : C_0\}$. Then

1. For each completion rule $\mathcal{R}$ that can be applied to S, and for each interpretation $\mathcal{I}$ we have $\mathcal{I}$ is a model of S iff $\mathcal{I}$ is a model of one of the systems S_i obtained by applying $\mathcal{R}$.
2. If S is a complete and clash-free constraint system, then S has a model.
3. If S contains a clash, then S does not have a model.
4. The completion algorithm terminates when applied to $\{x_0 : C_0\}$.

Remark 4.9 Indeed, Lemma 4.8 immediately implies decidability of satisfiability of $\mathcal{ALC}$-concepts. When the completion algorithm terminates, it stops with a tree whose leaves are all labelled with complete constraint systems. If C_0 is satisfiable, then $\{x_0 : C_0\}$ is also satisfiable, and thus one of the complete constraint systems is satisfiable by (1). By (3), this system must be clash-free. Conversely, if one of the complete constraint systems is clash-free, then it is satisfiable by (2), and because of (1) this implies that $\{x_0 : C_0\}$ is satisfiable. Hence the algorithm is a decision procedure for satisfiability of $\mathcal{ALC}$-concepts.

Lemma 4.8 can be proved as follows:

(1) is proved by investigating all rules separately. Since all constraint systems S_i generated by a rule are supersets of their ancestor S, it follows immediately that each model of S_i is also a model of S. The other direction depends on the respective rule, but is straightforward in most cases.

(2) can be proved by constructing a so-called *canonical model* for S from a clash-free, complete constraint system S. In Example 4.7, this was realized as follows: Variables in the constraint system became individuals of the canonical model, and concepts and roles were interpreted according to the constraint system. For other Description Logics, the construction of the canonical model can be more complicated.

(3) is straightforward. It suffices to show that no interpretation can satisfy a constraint system containing $\{x : A, x : \neg A\}$.

(4) can be proved by giving a *termination order*. This is a set $\mathcal{A}$ ordered by some well-founded order $\succ$. Well-foundedness ensures that there is no infinite chain $a_0 \succ a_1 \succ a_2 \succ \ldots$. Each constraint system S is associated to $\mathsf{size}(S) \in \mathcal{A}$ in such a way that, if S_i is obtained from S by application of one of the completion rules, then $\mathsf{size}(S) \succ \mathsf{size}(S_i)$. This property together with well-foundedness and the fact that each rule generates only finitely many constraint systems implies termination. For extensions of $\mathcal{ALC}$, this technique is applied in [Baader & Hanschke1991] and Section 6.1.2.

Intuitively, the following properties of the completion algorithm are responsible for its termination.

- All concepts of constraints added by the completion rules are *subconcepts* of the concept C_0 with which the algorithm started. It is not difficult to see that the number of subconcepts of a concept C is linear in the length of C.

- If a rule $\mathcal{R}$ can be applied to a constraint system S, then this is because of the presence of a particular constraint $a : C$ in S, and the application of $\mathcal{R}$ adds constraints whose concepts are strictly shorter than C.

- The introduction of role successors is always triggered by a constraint of the form $x : \exists R, C$, hence each variable has only finitely many role successors.

- Once a variable is introduced, it is present in all subsequent constraint systems, and constraints are never removed (these two properties prevent the algorithm from looping).

Although the completion algorithm is sound and complete, it is far from being optimal. For example, when started with the following concept C_n (whose length is quadratic in n), it generates a constraint system whose size is exponential in the length of C_n, i.e., the variables introduced by the algorithm together with the role successorship induced by constraints xRy build a binary tree of depth $n + 1$.

$$
\begin{aligned}
C_0 = \;& \exists R.A \sqcap \exists R.\neg A \sqcap \\
& \forall R.(\exists R.A \sqcap \exists R.\neg A) \sqcap \\
& \forall R.\forall R.(\exists R.A \sqcap \exists R.\neg A) \sqcap \\
& \forall R.\forall R.\forall R.(\exists R.A \sqcap \exists R.\neg A) \sqcap \\
& \forall R.\forall R.\forall R.\forall R.(\exists R.A \sqcap \exists R.\neg A) \sqcap \\
& \underbrace{\forall R. \ldots\ldots\ldots\ldots \forall R.}_{n \text{ times}}(\exists R.A \sqcap \exists R.\neg A)
\end{aligned}
$$

When started with C_n, the completion algorithm needs space exponential in the size of its input C_n, which is more than necessary: In [Schmidt-Schauß & Smolka1991; Donini *et al.*1991a], a different completion algorithm is presented which needs space that is only polynomial in the length of the input concept, and which works also for extensions of $\mathcal{ALC}$. Indeed, it was shown that this result is optimal since satisfiability of $\mathcal{ALC}$-concepts is PSPACE-complete.

Similar to the completion algorithm presented here, these PSPACE algorithms exploit the fact that for $\mathcal{ALC}$-concepts, it is sufficient to try to build a *tree model* for the input concept; see Section 4.5.3 for details. As a consequence of this fact, the variables of a constraint system together with the role successorship induced by constraints of the form xRy build a tree. In contrast to the completion algorithm presented here, the PSPACE algorithms handle the branches of this tree independently. Once a branch has been investigated, one

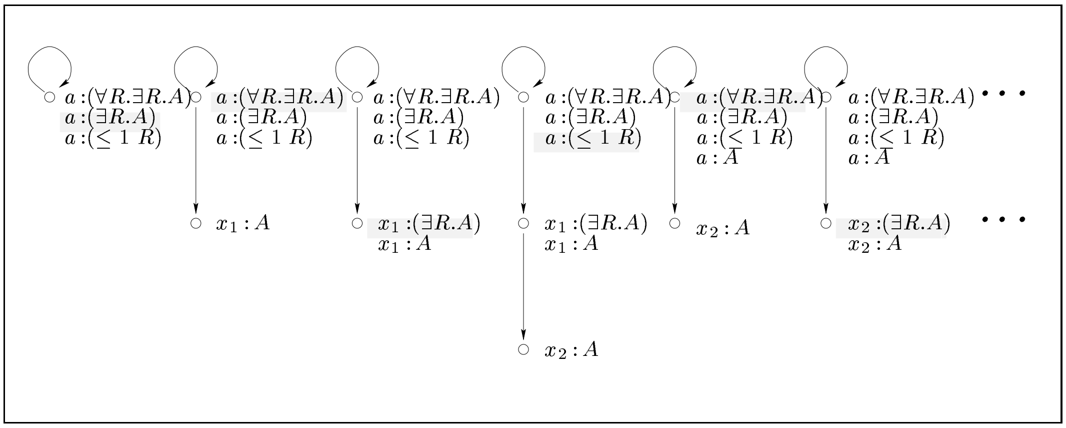

Figure 4.5: A non-terminating application of the completion rules.

can forget about it and reuse the space where it was stored–provided one keeps track of the results of these partial investigations.

As already mentioned at the end of Section 4.3, tableau-based algorithms like the completion algorithm presented above for $\mathcal{ALC}$ that decide satisfiability of concepts can be often modified such that they decide ABox consistency. To underline this fact, we have defined ABoxes and constraint systems in such a way that the latter are a special case of the former. In most cases, it suffices to define a *strategy* for the application of the completion rules in order to guarantee termination of the completion algorithm. A strategy is mainly an ordering in which the completion rules applied. For example, the following $\mathcal{ALC}$-ABox can lead to an infinite application of completion rules—if the rules are applied in the same ordering as in Figure 4.5. More precisely, the non-termination of this example is due to the fact that generating and identifying rules are applied in an arbitrary order. Termination for $\mathcal{ALC}$-ABoxes can be easily guaranteed by a simple strategy which prefers rules on "old" individuals to rules on individual variables introduced by the completion algorithms.

$$\begin{aligned}\mathcal{A} = \{\, & a:(\forall R.\exists R.A),\\ & a:(\exists R.A)\\ & a:(\leq 1\ R),\\ & aRa\}\end{aligned}$$

4.5 Extensions of $\mathcal{ALC}$

In this section, several known extensions of $\mathcal{ALC}$ are introduced. They will serve as a basis for further extensions which are described in the Chapters 6 and 7. The expressive power of $\mathcal{ALC}$ and these extensions are sketched in Section 4.5.3.

4.5.1 $\mathcal{ALC}$ extended by Complex Roles

Transitivity in Description Logics came (until recently, [Sattler1996]) always in the shape of the transitive closure of roles [Baader1991; Schild1991; De Giacomo & Lenzerini1995; Calvanese *et al.*1995]. The first extension of $\mathcal{ALC}$ by the transitive closure of roles was presented in [Baader1991]. The author presents the Description Logic $\mathcal{ALC}_{\mathsf{reg}}$, which extends $\mathcal{ALC}$ by allowing for regular expressions over role names in the place of role names. In [Baader1990a], a sound and complete tableau-based algorithm is given that decides satisfiability of $\mathcal{ALC}_{\mathsf{reg}}$-concepts. In the following, $\mathcal{ALC}_{\mathsf{reg}}$ as well as a restricted version of this logic, $\mathcal{ALC}_+$, is presented.

Definition 4.10 Starting with role names in N_R, *regular roles* are built using the role constructors composition ($R{\circ}S$), union ($R \sqcup S$), and transitive closure (R^+).

$\mathcal{ALC}_{\mathsf{reg}}$ is obtained from $\mathcal{ALC}$ by allowing, additionally, for regular roles inside concepts.

$\mathcal{ALC}_+$ is the extension of $\mathcal{ALC}$ obtained by allowing, additionally, for each role $R \in N_R$, the use of its transitive closure R^+ inside concepts.

The extension of the semantics to these complex roles is straightforward.

Definition 4.11 An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ of an $\mathcal{ALC}_{\mathsf{reg}}$-concept (resp. $\mathcal{ALC}_+$-concept) is an interpretation of $\mathcal{ALC}$-concepts that satisfies additionally the following equations:

$$\begin{aligned}
(R_1 \sqcup R_2)^{\mathcal{I}} &= {R_1}^{\mathcal{I}} \cup {R_2}^{\mathcal{I}},\\
(R_1 \circ R_2)^{\mathcal{I}} &= R_1^{\mathcal{I}} \circ R_2^{\mathcal{I}} = \{(d,f) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid \text{ There exists } e \in \Delta^{\mathcal{I}} \text{ with}\\
&\qquad\qquad (d,e) \in R_1^{\mathcal{I}} \wedge (e,f) \in R_2^{\mathcal{I}}\},\\
(R^+)^{\mathcal{I}} &= \cup_{i \geq 1} (R^{\mathcal{I}})^i, \text{ where } (R^{\mathcal{I}})^n = \underbrace{(R^{\mathcal{I}} \circ R^{\mathcal{I}} \circ \ldots \circ R^{\mathcal{I}})}_{n \text{ times}}.
\end{aligned}$$

For a regular role $\mathcal{R}$, an individual x said to be an $\mathcal{R}$-successor of an individual w in $\mathcal{I}$ iff $(w, x) \in \mathcal{R}^{\mathcal{I}}$.

Extending $\mathcal{ALC}$ by composition or disjunction of roles does not change the expressive power of $\mathcal{ALC}$ because of the following equivalences:

$$\begin{array}{lcllcl} \exists(R \circ S).C & \equiv & \exists R.(\exists S.C), & \exists(R \sqcup S).C & \equiv & (\exists R.C) \sqcup (\exists S.C), \\ \forall(R \circ S).C & \equiv & \forall R.(\forall S.C) \text{ and } & \forall(R \sqcup S).C & \equiv & (\forall R.C) \sqcap (\forall S.C). \end{array}$$

In contrast, extending $\mathcal{ALC}$ by the transitive closure of roles really increases its expressive power (see [Baader1990b] for a formal definition of expressive power). On one hand, $\mathcal{ALC}_+$- and $\mathcal{ALC}_{\mathsf{reg}}$-concepts cannot be translated to equivalent first-order formulae, whereas this is possible for $\mathcal{ALC}$-concepts; see Remark 4.3. This is due to the fact that the transitive closure of a relation cannot be expressed in first-order logic, in contrast to transitivity. On the other hand, $\mathcal{ALC}$ loses the *finite tree model property* when extended by the transitive closure of roles.[2]

As a consequence of this second fact, algorithms (such as tableau-based algorithms) that try to construct a model of a concept containing the transitive closure of roles need special "cycle detection mechanisms" [Baader1991] that prevent them from looping. For example, a naive modification of the completion algorithm for $\mathcal{ALC}$ presented in Section 4.4 would introduce an infinite number of R^+-successors when started with $x : C_0$ for

$$C_0 = A \sqcap (\exists R.A) \sqcap (\forall R^+.\exists R.A).$$

A terminating modification of this algorithm must notice that all these R^+-successors involve the same constraints and then stop the generation of new successors. Moreover, this modification must recognise cases where the satisfaction of constraints is postponed in each step. This could happen, for example, while trying to construct a model of

$$C_1 = A \sqcap (\exists R^+.\neg A) \sqcap (\forall R^+.A).$$

A naive modification of the completion algorithm for $\mathcal{ALC}$ would start with $\{x_0 : C_0\}$, and would run into an infinite loop while trying to generate an R^+-successor of x_0 in $\neg A$. This is due to the fact that the constraint $x_0 :(\exists R^+.\neg A)$ can be satisfied by any R^i-successor of x_0, and that the length i of this R-chain

[2] A Description Logic has the finite tree model property if each satisfiable concept has also a model that has the shape of a finite tree; see Section 4.5.3 for details.

is not fixed or bounded by C_0 in an obvious way. Summing up, a sound and complete modification of the completion algorithm for $\mathcal{ALC}$ has to distinguish between cases where constraints on individuals propagated along some (possibly infinite) role chain are simply re-generated but satisfied (as in the example with C_0) and cases where the satisfaction of constraints is always postponed (as in the example with C_1).

Unfortunately, these cycle detection mechanisms need to store a large amount of information and this cannot be accomplished using polynomial space: As stated in [Fischer & Ladner1979; Pratt1979], satisfiability of Propositional Dynamic Logic (PDL) formulae is ExpTime-complete. A translation of this result from PDL to the vocabulary of Description Logics, namely to $\mathcal{ALC}_{\mathsf{reg}}$, can be found in [Schild1991]. This translation is, beside others, satisfiability preserving. That is, a PDL formula ϕ is satisfiable if and only if its translation into an $\mathcal{ALC}_{\mathsf{reg}}$-concept C_ϕ is satisfiable. Since $\mathcal{ALC}_{\mathsf{reg}}$ is propositionally closed, subsumption can be reduced to satisfiability and vice versa, and, since these reduction are linear, subsumption of $\mathcal{ALC}_{\mathsf{reg}}$-concepts is also ExpTime-complete. Finally, the hardness proof in [Fischer & Ladner1979] does not make any use of composition or disjunction of roles, hence $\mathcal{ALC}_+$ is also ExpTime-complete.

4.5.2 $\mathcal{ALC}$ extended by Number Restrictions

Since objects are often characterised by the number of objects they are related to via some relation, number restrictions are present in almost all implemented systems [Brachman & Schmolze1985; Patel-Schneider *et al.*1991; Peltason1991; MacGregor & Brill1992; Baader *et al.*1994]. They occur in their most simple form in $\mathcal{ALCN}$, an extension of $\mathcal{ALC}$ introduced in [Hollunder *et al.*1990; Donini *et al.*1991a]. $\mathcal{ALCN}$ is obtained from $\mathcal{ALC}$ by allowing, additionally, for constructs which describe, for example, that an individual has *at least 7* parts or *at most 4* interfaces.

Definition 4.12 $\mathcal{ALCN}$ (resp. $\mathcal{ALC}_+\mathcal{N}$) is obtained from $\mathcal{ALC}$ (resp. $\mathcal{ALC}_+$) by allowing, additionally, for concepts of the form $(\leq n\ R)$ and $(\geq n'\ R)$, where n, n' are nonnegative integers and R is a role name.

An *interpretation* $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ of an $\mathcal{ALCN}$-concept (resp. $\mathcal{ALC}_+\mathcal{N}$-concept) is an interpretation of $\mathcal{ALC}$-concepts that additionally satisfies the following equations:

$$\begin{aligned}(\geq n\ R)^\mathcal{I} &= \{d \in \Delta^\mathcal{I} \mid \#\{e \in \Delta^\mathcal{I} \mid (d,e) \in R^\mathcal{I}\} \geq n\},\\ (\leq n\ R)^\mathcal{I} &= \{d \in \Delta^\mathcal{I} \mid \#\{e \in \Delta^\mathcal{I} \mid (d,e) \in R^\mathcal{I}\} \leq n\},\end{aligned}$$

where $\#M$ denotes the cardinality of a set M.

Similar, $\mathcal{ALC}_{\mathsf{reg}}\mathcal{N}$ and $\mathcal{ALC}_{+}\mathcal{N}$ denote the extensions of $\mathcal{ALC}_{\mathsf{reg}}$ and $\mathcal{ALC}_{+}$ by number restrictions on role names. This is to say that in $\mathcal{ALC}_{\mathsf{reg}}\mathcal{N}$ and $\mathcal{ALC}_{+}\mathcal{N}$, the complex roles available in $\mathcal{ALC}_{\mathsf{reg}}$ and $\mathcal{ALC}_{+}$ are not allowed inside number restrictions—here, only role names can be used.

Since $\mathcal{ALC}$ is propositionally closed, we can express all relations in $\{=,<,>\}$ inside number restrictions, for example $(> n\ R) \equiv \neg(\leq n\ R)$ and $(= n\ R) \equiv ((\leq n\ R) \sqcap (\geq n\ R))$.

As shown in [Donini *et al.*1991a], adding number restrictions to $\mathcal{ALC}$ does not really increase its computational complexity: Satisfiability and subsumption of concepts are PSPACE-complete.

In [Hollunder & Baader1991], a more expressive variant of number restrictions, namely so-called *qualifying* number restrictions are presented and investigated. Qualifying number restrictions are of the form $(\leq n\ R.C)$ or $(\geq n\ R.C)$, and they are interpreted as the set of those individuals having at least (resp. at most) n R-successors that are instances of C, i.e.,

$$\begin{array}{rcl}(\geq n\ R.C)^{\mathcal{I}} & = & \{d \in \Delta^{\mathcal{I}} \mid \#\{e \in \Delta^{\mathcal{I}} \mid (d,e) \in R^{\mathcal{I}}\} \cap C^{\mathcal{I}} \geq n\}, \\ (\leq n\ R.C)^{\mathcal{I}} & = & \{d \in \Delta^{\mathcal{I}} \mid \#\{e \in \Delta^{\mathcal{I}} \mid (d,e) \in R^{\mathcal{I}}\} \cap C^{\mathcal{I}} \leq n\}.\end{array}$$

As a consequence of the tableau-based algorithm presented in [Hollunder & Baader1991], adding qualifying number restrictions to $\mathcal{ALC}$ preserves the finite-tree-model property.

4.5.3 Expressive power of the basic Description Logics

In general, logics can be classified with respect to the form the models of satisfiable formulae have. Considering the form that models of satisfiable concepts of a certain Description Logic have is useful because this form is closely related to its expressive power. For example, if a Description Logic has the finite-model property, then no concept can be built within this logic which enforces an infinite model.

More precisely, each satisfiable concept of a logic having the *finite-model property* has a model with finite interpretation domain. Given that model checking is decidable, the finite model property implies that the set of all satisfiable concepts is recursively enumerable. Furthermore, most Description Logics can be viewed as fragments of first-order logic (see Remark 4.3 for a translation from $\mathcal{ALC}$ to first-order logic), for which unsatisfiable formulae are known to be recursively enumerable. Together, this yields decidability of satisfiability of these logics: Deciding whether a concept C is satisfiable can be done by first translating C into a corresponding first-order formula $\phi_C(x)$, and then testing in parallel whether $\phi_C(x)$ is unsatisfiable and whether interpretations $\mathcal{I}_j$ are models of $\phi_C(x)$. Given that these interpretations are completely enumerated from small ones to larger ones, this yields a decision procedure for (Description) logics having the finite-model property. However, in general this procedure is far from being optimal and does not provide any upper bounds for the computational complexity of the satisfiability problem under consideration.

A language not having the finite-model property is not necessarily undecidable (see [Grädel *et al.*1997] for example), but a lack of this property is a good hint that reasoning might be very difficult. In fact, all Description Logics investigated in this thesis which lack the finite-model property turn out to have undecidable satisfiability problems.

Satisfiable concepts of a Description Logic having the *tree-model property* have some model in the form of a tree: If C is satisfiable, then it has a model $\mathcal{I}$ such that

- there is some $x_0 \in C^{\mathcal{I}}$ that has no R-predecessor in $\Delta^{\mathcal{I}}$ for all roles R,
- each element of $\Delta^{\mathcal{I}}$ can be reached from x_0 via a role chain[3], and
- each $y \neq x_0$ in $\Delta^{\mathcal{I}}$ is an R-successor of exactly one $y' \in \Delta^{\mathcal{I}}$ for one role name R.

Finally, if a Description Logic has the *finite-tree-model property*, then each satisfiable concept has a finite tree model.

In contrast to $\mathcal{ALC}_+$, $\mathcal{ALCN}$ has the *finite-tree-model property*, which will not be lost even when extended by allowing for *qualified* number restrictions. For $\mathcal{ALC}_+$, the possibility to use the transitive closure of a role inside value restrictions is the reason for the lack of the finite-tree-model property: For example, the following concept describes instances of A having some R-successor in A and where each individual reachable over some R-path has itself some R-successor in A:

$$A \sqcap (\exists R.A) \sqcap (\forall R^+.(\exists R.A)).$$

Obviously, this concepts is satisfiable, but each of its models has either an infinite R-chain or it contains some R-cycle. Nevertheless, $\mathcal{ALC}_+$ still has the

[3]Role chains are roles of the form $R_1 \circ \ldots \circ R_n$.

tree-model property, which means that each satisfiable $\mathcal{ALC}_+$-concept has a (not necessarily finite) tree model.

The expressive power added by the transitive closure to $\mathcal{ALC}$ becomes also apparent in the following example:

$$\texttt{Device} \sqcap (\exists \texttt{has\_part}^+.(\exists \texttt{is\_made\_of}.\texttt{Carcinogenic}). \tag{4.1}$$

Using the transitive closure of `has_part`, one can refer to parts at some level of decomposition not known in advance, without restricting this level a priori. If this maximum level could be restricted to, say, n, those dangerous devices can be described by

$$\begin{array}{l}(\exists \texttt{has\_part}.\exists \texttt{is\_made\_of}.\texttt{Carcinogenic}) \sqcup \\ (\exists \texttt{has\_part}.\exists \texttt{has\_part}.\exists \texttt{is\_made\_of}.\texttt{Carcinogenic}) \sqcup \\ (\exists \texttt{has\_part}.\exists \texttt{has\_part}.\exists \texttt{has\_part}.\exists \texttt{is\_made\_of}.\texttt{Carcinogenic}) \sqcup \ldots \\ (\underbrace{\exists \texttt{has\_part}.\ldots \exists \texttt{has\_part}.}_{n \text{ times}} \exists \texttt{is\_made\_of}.\texttt{Carcinogenic})\end{array}$$

Unfortunately, beside the necessity of fixing an upper bound n, a large number of disjunctions over complex concepts is involved which leads, in general, to performance degradation of the inference algorithm. One can think of optimising techniques which exploit the fact that these disjuncts are of a similar structure, but it is probably more efficient to directly use the transitive closure of roles together with specific optimisation techniques.

When constructing an algorithm for testing satisfiability of concepts, tree-model properties turn out to be very useful. Tableau-based algorithms, in general, test satisfiability by trying to build a canonical model. For logics having the (finite-)tree-model property, one has only to consider models having a (finite) tree structure which makes the design of these algorithms easier: For most of these logics, the attempt can be stopped when a tree is constructed whose depth is (polynomially or exponentially) bounded by the length of the input concept. Furthermore, it is easy to avoid the algorithm from looping and thus to ensure termination while investigating tree structures–given that the outdegree can be bounded, which is usually the case. In arbitrary structures, special cycle detection techniques have to be applied to yield termination. Additionally, to proof the algorithm's soundness is mostly easy if the canonical model constructed for a satisfiable concept C_0 has the form of a finite tree: By induction on the depth of this tree, it can be shown that the canonical model satisfies all constraint on variables at this depth. In the presence of cycles or infinite chains, this is no longer possible, and the proof might become more difficult.

Summing up, these model properties strongly influence the reasoning techniques one has to apply. As a rule of thumb, reasoning in tree models is mostly easier than in arbitrary ones, and reasoning in finite models is mostly easier than in possibly infinite ones. On the other hand, these model properties are closely related to the expressive power of a logic. For example, using a Description Logic with the tree-model property, it is impossible to describe devices where all those devices it is connected to are controlled by the same control unit (see Chapter 1, concept 1.1), and that are connected to at least two devices. A concept describing devices with these properties has clearly no tree model since such a device is always related to a control unit via at least two different paths. For an application where it is crucial to describe and reason about similar concepts, the expressive power of a Description Logic having the tree-model property is not adequate. Similarly, if the application asks for concepts describing infinite structures only, a Description Logic having the finite model property is not expressive enough.

Chapter 5

Prototype Implementation

In this chapter, the integration of the DL system CRACK [Bresciani *et al.*1995] into the process modeling tool MODKIT [Bogusch *et al.*1996] is motivated and described. This integration is a part of the cooperation between the Lehrstuhl für Prozeßtechnik and the Lehr- und Forschungsgebiet Theoretische Informatik of RWTH Aachen and was realised with the support and involvement of R. Bogusch and W. Geffers.

5.1 The modeling tool MODKIT

MODKIT was developed at the Lehrstuhl für Prozeßtechnik at the University of Technology in Aachen in order to empirically evaluate the ideas formalised within VEDA [Baumeister *et al.*1998]. It is implemented in G2, which is a tool for developing and executing real-time expert systems [Gen1995]. G2 was originally applied to process monitoring and control tasks within the chemical and manufacturing industries. Its flexible development environment has led to many other application domains, not all of which are centred around real-time processing.

G2 allows for the definition of classes, rules, and procedures activated by the user or triggered by particular events. However, despite the fact that G2 allows for multiple inheritance,[1] the possibilities to describe classes, respectively the objects belonging to these classes, are rather restricted. For example, it is not possible to refine the range of an attribute whose range is already specified in a

[1] This can be compared with the possibility to define, in Description Logics, one concept as a subconcept of more than one concept defined in the TBox.

superclass. As a consequence, the class hierarchy in ModKit is poorly structured in that the description of a class contains mostly only the name of the according superclasses, and no further information about what distinguishes the instances of this class from the instances of its superclasses.

Apart from this weakness, process modeling using ModKit can be said to be rather comfortable. In general, setting up a model in ModKit works as follows. First, the structure of the model is specified by choosing devices and connections from a catalogue; see Figure 5.1. This is done by dragging and dropping items representing devices or connections from the catalogue into the current workspace. Next, these items are connected using predefined connection posts, and their properties and behaviour is specified using the dialogues built for these purposes; see Figure 5.2. If composed items were chosen, their components are specified in a way similar to the one that is trodden to specify a whole model. This procedure is strongly supported by online help, and the documentation of the decisions taken and the choices made during this procedure is supported by predefined forms attached to the objects representing devices and connections. Once the model is completely specified, the simulation of its behaviour can easily be realised due to the integrated simulation tools and source code generators. The latter automatically generate the source code for the former from the specified model. Finally, the whole process of setting up a model is supported by the formalisation of the actions undertaken on the way towards a process model [Lohmann1998; Krobb1997]. However, the design of a model is a highly creative process, and can thus only partially be guided.

5.2 The terminological knowledge representation system Crack

Crack, implemented at the Istituto per la Ricerca Scientifica e Tecnologica (IRST) in Trento, Italy [Bresciani *et al.*1995], is a DL system whose development was driven by the need for a DL system

- that implements a Description Logic with high expressive power, namely an extension of $\mathcal{ALC}$ by features (i.e., functional roles, which are a simple form of number restrictions), inverse roles, the intersection of roles, and individuals (which allow the user to enumerate concrete individuals and state, for example, that objects are related to one of these via a specific role by ($\exists$`connected-to`.(one_of {`valve7`, `valve9`, `valve17`}))),

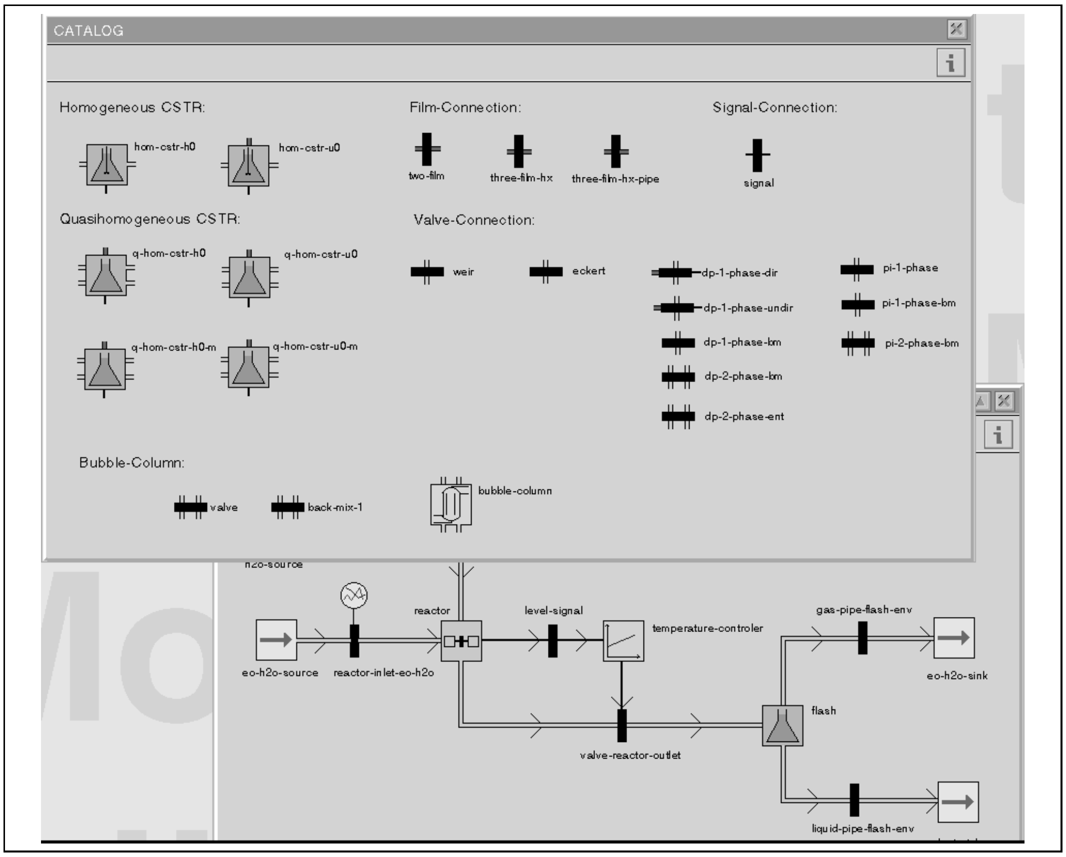

Figure 5.1: A MODKIT screenshot showing the catalogue and the structure of the ethylene-glycol process.

- whose algorithms are based on sound and complete tableau-based algorithms, and
- that is modular, which means that it can easily be extended to more expressive Description Logics, namely those encompassing new concept- or role-forming constructors.

The implementation of CRACK had to cope with two challenges: On the one hand, the goal of being modular can only be achieved by a rather direct implementation of the completion rules of the tableau algorithm—which were not designed for implementation purposes, but for proof purposes. On the other hand, the implementation has to be highly optimised—otherwise, the high computational complexity of the underlying Description Logic would prevent any realistic application. Fortunately, the implementation reached both goals:

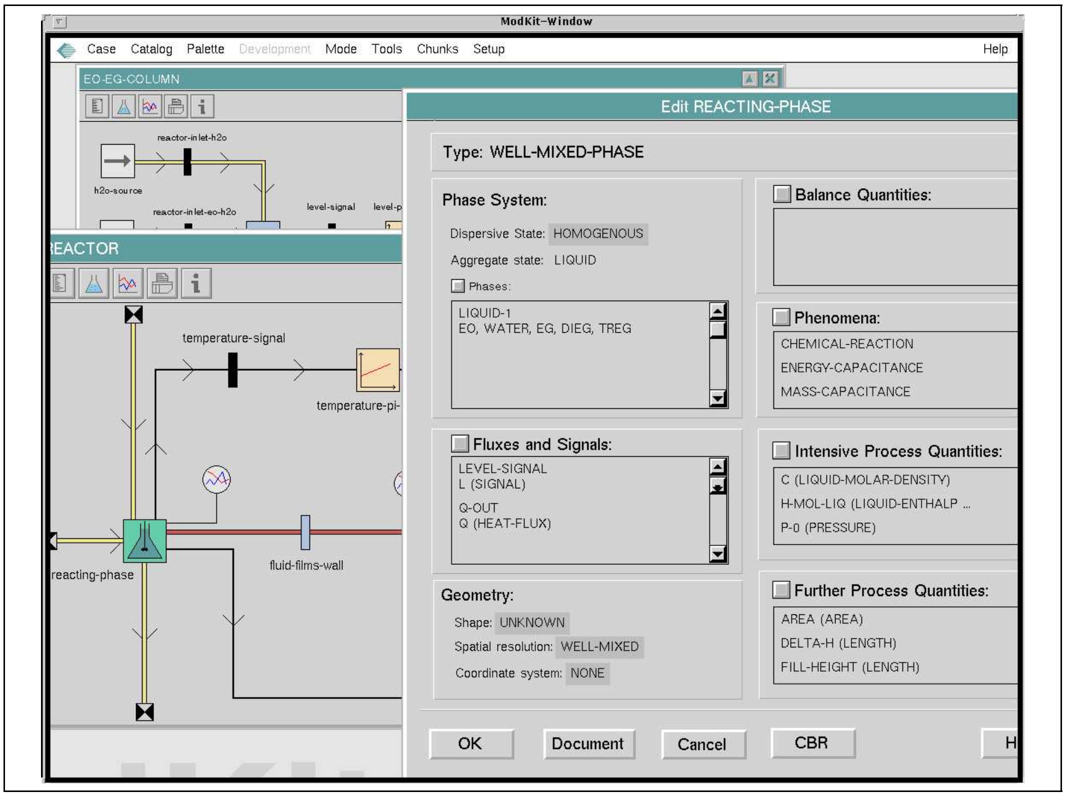

Figure 5.2: A ModKit screenshot showing the dialogue for the property specification of a reactor.

The optimisation techniques applied proved to be so efficient that even large knowledge bases can be classified within an acceptable amount of time.

The authors of Crack and Fact [Horrocks & Gough1997] are working on a descendant of both Crack and Fact. Fact is an implementation of an extension of $\mathcal{ALC}_{R^+}$ introduced in Chapter 7 which was motivated by the promising results concerning the computational complexity described there. We hope to be able to replace Crack by this new DL system because it will be more appropriate for the representation of aggregated objects; see the discussion at the end of Section 7.4.

5.3 The integration of Crack into ModKit

The ideas concerning the integration of Crack into ModKit are results of a fruitful cooperation with R. Bogusch. In order to use the system services provided by CRACK for a better structuring of the classes and objects in MODKIT, these classes must be translated into CRACK-concepts. Given that the engineer should be supported when defining new classes, this translation has to be generated automatically. In Figure 5.3, the architecture of the integration is presented. The *translator* is a MODKIT program which realises the above mentioned translation of MODKIT class descriptions into concepts in CRACK syntax. For the translation of a MODKIT class C, besides the information specified in the definition of C explicitly, the translator takes also into account information concerning instances of C that is available elsewhere. For example, if a MODKIT relation R is defined in such a way that its domain is C and its range is another class D, then the translator explicitly states in the translation of C that all objects related to instances of C via the relation R are instances of D. While realising the integration, the MODKIT class definitions were modified in such a way that the MODKIT class hierarchy and the taxonomy of the corresponding concepts agree. This could be realised rather easily because the number of classes was still small. In the following, we assume that, for already defined classes, its place in the MODKIT hierarchy is identical to the place of its corresponding CRACK concept in the taxonomy.

The communication between MODKIT and CRACK is realised by a *bridge process* which was implemented in C by A. Beber of IRST in Trento, Italy, when he was a guest at the Lehrstuhl für Prozeßtechnik of RWTH Aachen. The bridge process is invoked via the GSI interface of MODKIT using Unix sockets. The bridge process and MODKIT communicate via the GSI interface of MODKIT. This communication as well as the tailoring of the bridge process to the specific requirements of the overall integration was accomplished by W. Geffers of the Lehrstuhl für Prozeßtechnik.

For modularity reasons, CRACK is not launched automatically together with MODKIT. CRACK is launched and connected to MODKIT by simply clicking on a MODKIT action button, and then the basic system services are called across the bridge process. These system services are called automatically from built-in routines in order to support the engineer in structuring both MODKIT classes as well as MODKIT individuals.

MODKIT classes have an attribute `crack-description` whose value, a string, is generated automatically and set by the translator. The way actions trigger the invocation of the translator guarantees that the translation of each class is up-to-date before CRACK system services are invoked. The attribute `crack-description` contains, in CRACK syntax, all the information concerning the corresponding class that are available from MODKIT. As a matter of fact, this description cannot contain any knowledge concerning triggers, rules

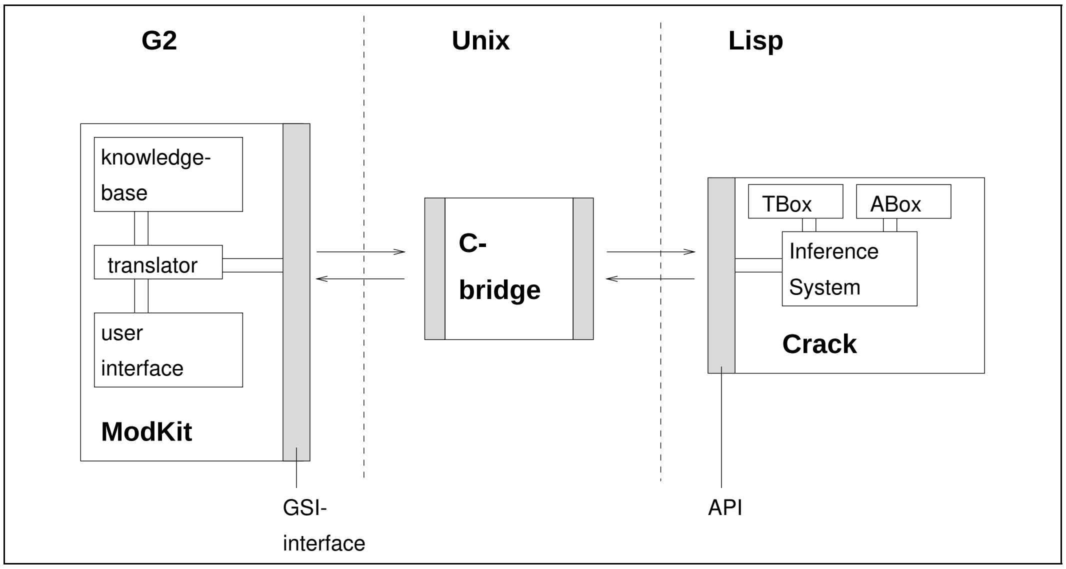

Figure 5.3: The architecture of the integration of CRACK into MODKIT.

or procedures associated to a class. Hence the information concerning a class that is visible to CRACK is incomplete. As a consequence, no MODKIT class will be migrated in the class hierarchy automatically because CRACK inferred that the corresponding concept has a place in the taxonomy that is different from the place of the corresponding MODKIT class.

The answers given by CRACK are used as follows: The simplest way in which CRACK can be used is the visualisation of the taxonomy of the concepts that are translations of MODKIT classes. More sophisticated uses of CRACK's system services are described in the following.

Modifying and reclassifying prototypes

When creating models using MODKIT, a common technique is to copy an object o (a so-called prototype) representing a device or a connection from the catalogue or from another model, and to modify it according to the specific needs of the current model. If o was instance of class C when it was copied, it continues to be an instance of C even if its properties are completely changed. In MODKIT, like in many object-oriented systems, an object continues being an instance of the class it was instantiated from until this relationship is changed by hand. For example, when looking more closely at models that were created using MODKIT, we found an atomic device that was modified in such a way that it had parts. For a better reuse of already defined objects and for a better cooperation among several engineers, an object o should always be an instance of one of the classes it "fits best". The meaning of "fitting best" is explained in the following. We assume that o's properties are possibly partially given, that is, o has all the properties that are stated explicitly, plus possibly other properties. Then a class C is said to "fit" if o could be an instance of C without causing inconsistencies whatever additional properties o might have. A class C is said to "fit best" an object o if no class that is more specific than C also fits o. For one object o, there might be several classes that fit best o. Like other object-oriented systems, MODKIT allows only for single instantiation, that is, each object is an instance of exactly one class. As stated above, an object o is not necessarily an instance of some class it fits best, and it would already be helpful if this could be ensured.

However, if the migration of objects to instances of classes they fit best is realised by the engineer, it becomes costly, possibly incorrect, and can cause inconsistencies. It can be supported by applying the system services of CRACK, namely the retrieval of the most specific concepts for an object. The integration of CRACK into MODKIT supports this migration as follows: Each time the properties of an object are changed, CRACK is automatically asked to compute the most specific concepts the translation of this object is an instance of. The corresponding classes are recommended to the user to migrate this object to. He or she can then choose one of these classes as the new class the object under consideration is an instance of.

Defining a new class

For the definition of a new class O, CRACK is used as follows. Roughly spoken, the user sketches the class definition in a dialogue designed for this purpose. Then this class definition is automatically translated into a CRACK concept and classified into the already existing taxonomy. The results of this classification are shown to the user, which can then investigate the subsumers and subsumees of O's translation and change the definition of the class if its place in the taxonomy does not match his or her intuition. Only when O's place in the taxonomy is accepted by the user, O is added as a new class, having the subsumers and subsumees proposed by CRACKas super- and subclasses. In this way, MODKIT's class hierarchy and the corresponding taxonomy are always kept identical.

5.4 Experiences

Following the argumentation of this thesis, the expressive power of Crack is not high enough for this application. However, Crack was the most expressive DL system with sound and complete inference algorithms available at the time the integration started. Fortunately, all automatically deducible information concerning ModKit classes and objects can be expressed in Crack. Hence even a more sophisticated translator would only output concepts that can be handled by Crack. On the other hand, there are properties of classes and objects that cannot be deduced by any translator (for example those implied by triggers or rules). As a consequence, no automatic changes of the class hierarchy or objects are triggered by answers of Crack.

The first positive aspect of this integration showed up while realising the integration: The ModKit class hierarchy, which has grown constantly over the last years, was examined in detail. Inconsistencies were detected and improvements towards a better, deeper structure of this hierarchy were made. This included the modification of ModKit class descriptions: When the integration was started, there were many classes for which the only information present in ModKit was the set of their super- and subclasses. The motivation for the introduction and the meaning of these classes could only be guessed from their names. Translating these classes and arranging the corresponding concepts with respect to the subsumption relation revealed this lack of description, led to discussions among the process engineers, and to a better, declarative description of these classes. An obvious advantage of these modifications is that new engineers working with ModKit can understand more easily the class hierarchy.

The second positive effect is due to the support the integration of Crack into ModKit provides for the migration of objects and classes. The system services provided by Crack guide the user in the migration of objects whose properties were modified into classes that fit best, and in the definition of new classes for new types of objects. This "cleaning up" has still to be done by hand by the engineers, but they are now supported by the system services of Crack. This supports both increases the motivation of the engineers to do this cleaning up as well as decreases the risk of senseless migrations or inconsistencies.

However, the system services available in current DL system can still be extended to provide an even better support: For example, given a set of objects, a system service that automatically generates a class description for a possibly new class that fits best this set would be very useful. Furthermore, a system service that, given a set of objects, partitions this set into subsets of objects which are relatively similar to each other would be useful: It could be used to determine sets of objects for which it would be useful to introduce new classes. These system services are not yet available in current DL systems. To provide these services, a precise definition of their functionality is required, and adequate algorithms have to be developed. These investigations will be part of future work carried out in cooperation between the Lehr- und Forschungsgebiet Theoretische Informatik and the Lehrstuhl für Prozeßtechnik at the University of Technology in Aachen.

Chapter 6

Extending Description Logics by Expressive Number Restrictions

There are two ways in which we increase the expressive power of traditional number restrictions in Description Logics: First, we allow the use of *complex roles* inside number restrictions. These constructors enable us, for example, to express statements such as "for this object, we model at least 4 phenomena that are also modeled for its neighbours" by using the number restriction $(\geq 4\ ((\texttt{connected-to} \circ \texttt{has-phenomena}) \sqcap \texttt{has-phenomena}))$. In this example, composition ($\circ$) and conjunction ($\sqcap$) of roles are used inside a number restriction. Other role-forming constructors investigated in this chapter are union ($\sqcup$) and inversion ($\cdot^{-1}$). Second, in *symbolic* number restrictions, numerical variables can be used in the place of nonnegative integers. Thus, one has no longer to fix a number for the (lower or upper) bound of role successors, but one can express statements such as "having *the same number* of `inputs` and `outputs`" or "having *less* `outputs` than the objects `connected-to` it have `inputs`".

Before investigating each of these Description Logics in detail, these extensions of $\mathcal{ALCN}$ (resp. $\mathcal{ALC}_{\mathsf{reg}}\mathcal{N}$ and $\mathcal{ALC}_{+}\mathcal{N}$) are introduced and their expressive power is highlighted by some examples. In the following sections, the computational complexity of the corresponding inference problems is investigated.

$\mathcal{ALC}$ extended by Number Restrictions on Complex Roles

Several extensions of $\mathcal{ALCN}$ and $\mathcal{ALC}_{+}\mathcal{N}$ obtained by allowing different kinds of complex roles inside number restrictions will be investigated, and we start by introducing a scheme for building these extensions. The name of such a scheme consists of the name of the basic language followed by the set of role constructors that are allowed inside number restrictions.

Definition 6.1 Starting with *atomic roles* from a set N_R of role names, *complex roles* are built using the role constructors composition ($R \circ S$), union ($R \sqcup S$), intersection ($R \sqcap S$), and inversion (R^{-1}).

For a set $M \subseteq \{\sqcup, \sqcap, \circ, {}^{-1}\}$ of role constructors, we call a complex role R an *M-role* iff R is built using only constructors from M.

The set of *$\mathcal{ALCN}(M)$-concepts* (resp. *$\mathcal{ALC}_+\mathcal{N}(M)$-concepts* and *$\mathcal{ALC}_{\mathsf{reg}}\mathcal{N}(M)$-concepts*) is obtained from $\mathcal{ALCN}$-concepts (resp. $\mathcal{ALC}_+\mathcal{N}$- and $\mathcal{ALC}_{\mathsf{reg}}$-concepts) by additionally allowing for M-roles inside number restrictions.

An *interpretation* $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ of an $\mathcal{ALCN}(M)$ -concept (resp. $\mathcal{ALC}_+\mathcal{N}(M)$ -concept) is an interpretation of $\mathcal{ALC}_+\mathcal{N}$-concepts that satisfies additionally the following equations:

$$
\begin{aligned}
(R_1 \sqcup R_2)^\mathcal{I} &= {R_1}^\mathcal{I} \cup {R_2}^\mathcal{I},\\
(R_1 \sqcap R_2)^\mathcal{I} &= {R_1}^\mathcal{I} \cap {R_2}^\mathcal{I},\\
(R_1 \circ R_2)^\mathcal{I} &= \{(d,f) \in \Delta^\mathcal{I} \times \Delta^\mathcal{I} \mid \exists e \in \Delta^\mathcal{I} : (d,e) \in R_1^\mathcal{I} \wedge (e,f) \in R_2^\mathcal{I}\},\\
(R^{-1})^\mathcal{I} &= \{(e,d) \in \Delta^\mathcal{I} \times \Delta^\mathcal{I} \mid (d,e) \in R^\mathcal{I}\}.
\end{aligned}
$$

Composition is present in all extensions investigated in this thesis: On the one hand, composition strongly increases the expressive power in that it allows to restrict the number of role-*chain*-successors. In simple cases, this can also be realized using pure $\mathcal{ALC}$. For example, using three disjoint concepts A_1, A_2, and A_3, the concept $(\geq 3\ R{\circ}S)$ can be rewritten as

$$(\exists R.\exists S.A_1) \sqcap (\exists R.\exists S.A_2) \sqcap (\exists R.\exists S.A_3)$$

However, this does not always work: $\mathcal{ALCN}$ has the tree-model property, even when extended by qualifying number restrictions. As a consequence, $\mathcal{ALCN}$ is not able to express upper bounds for the number of role-chain-successors, thus enforcing intermediate successors to share successors, which can easily be expressed using $\mathcal{ALCN}(\circ)$. For example, the concept

$$D = (\geq 2\ R) \sqcap (\forall R.(\geq 1\ S)) \sqcap (\leq 1\ R{\circ}S)$$

is obviously satisfiable, but each of its instances x has an $R{\circ}S$-successor which is connected to x via at least 2 different role paths. This example shows that $\mathcal{ALCN}(\circ)$ does not have the tree-model property. Furthermore, $\mathcal{ALCN}(\circ)$ can be said to be more expressive than $\mathcal{ALCN}$ because the fact that $\mathcal{ALCN}$ has the tree-model property implies that no $\mathcal{ALCN}$-concept is equivalent to D.

This kind of expressive power added by composition in number restrictions is of strong interest for the process engineering application because the devices and connections of a process model are, in general, not interrelated in a tree-like way but in more complex structures. Hence the possibility to describe non-tree structures is important for this application. Since composition in number restrictions provides this possibility, this constructor will be investigated in detail while looking for an Description Logic adequate for the representation of the standard building blocks of process models.

On the other hand, decidability results for $\mathcal{ALCN}(M)$ without composition follows immediately from results in [Grädel *et al.*1997]. They are discussed in Section 6.3.

Intersection on roles inside number restrictions is needed if one wants to express, for example, that a device is powered by something it is connected to, i.e.

$$\texttt{Device} \sqcap (\geq 1\ \texttt{powered-by} \sqcap \texttt{connected-to}).$$

Intersection in combination with **composition** comes in, for example, if we want to describe devices that share at least 1 power supply with devices they are connected to, i.e.

$$\texttt{Device} \sqcap (\geq 1\ \texttt{powered-by} \sqcap (\texttt{connected-to} \circ \texttt{powered-by})).$$

Union of roles is needed to restrict the total number of individuals that are related via several, not necessarily disjoint roles. For example, `connected-to` and `has-part` are not necessarily disjoint roles, and union can be used to describe complex devices which are connected to or have as a part at least, say, 7 objects, i.e.

$$\texttt{Device} \sqcap (\geq 7\ \texttt{connected-to} \sqcup \texttt{has-part}).$$

The above concept could be rewritten by a disjunction of concepts involving intersection of roles inside number restrictions, but only in a very complicated concept, i.e.,

$$
\begin{array}{l}
\texttt{Device} \sqcap (\geq 7\ \texttt{has-part}) \sqcup \\
((\geq 1\ \texttt{connected-to}) \sqcap (\geq 6\ \texttt{has-part}) \sqcap (\leq 0\ \texttt{has-part} \sqcap \texttt{connected-to})) \sqcup \\
((\geq 2\ \texttt{connected-to}) \sqcap (\geq 5\ \texttt{has-part}) \sqcap (\leq 0\ \texttt{has-part} \sqcap \texttt{connected-to})) \sqcup \\
\ldots \\
((\geq 2\ \texttt{connected-to}) \sqcap (\geq 6\ \texttt{has-part}) \sqcap (\leq 1\ \texttt{has-part} \sqcap \texttt{connected-to})) \sqcup \\
((\geq 3\ \texttt{connected-to}) \sqcap (\geq 5\ \texttt{has-part}) \sqcap (\leq 1\ \texttt{has-part} \sqcap \texttt{connected-to})) \sqcup \\
\ldots \\
((\geq 6\ \texttt{connected-to}) \sqcap (\geq 6\ \texttt{has-part}) \sqcap (\leq 5\ \texttt{has-part} \sqcap \texttt{connected-to})) \sqcup \\
(\geq 7\ \texttt{connected-to}),
\end{array}
$$

Apart from the unreadability of the above concept, there is another argument against the use of intersection instead of union inside number restrictions: As we will see later, intersection leads to an even higher complexity than union.

Union in combination with **composition** comes in, for example, if we want to describe a loosely connected part of a model. The following concept describes a device that belongs to such a loosely connected part:

$$
\begin{array}{ll}
\texttt{Device} \sqcap (\leq 6 & \texttt{connected-to} \sqcup \\
& (\texttt{connected-to} \circ \texttt{connected-to}) \sqcup \\
& (\texttt{connected-to} \circ \texttt{connected-to} \circ \texttt{connected-to})).
\end{array}
$$

Inversion of roles can be used to express, for example, that a power supply supplies power for more than one device, i.e.

$$\texttt{Power-supply} \sqcap (\geq 2\ \texttt{powered-by}^{-1}).$$

Inversion in combination with **composition** is needed, for example, if we want to express that all parts of a device belong exclusively to this device,

$$\texttt{Device} \sqcap (\leq 1\ \texttt{has-part} \circ \texttt{has-part}^{-1}).$$

In Section 6.1, the complexity of satisfiability and subsumption of these extensions will be investigated.

$\mathcal{ALC}$ extended by Symbolic Number Restrictions

Definition 6.2 Let $N_V = \{\alpha, \beta, \ldots\}$ be a set of numerical variables. Then $\mathcal{ALCN}^S$ is obtained from $\mathcal{ALCN}$ by additionally allowing for

- *symbolic* number restrictions $(\leq \alpha\ R)$ and $(\geq \alpha\ R)$ for a role name R and a numerical variable α, and
- the existential quantification $(\downarrow\alpha.C)$ of numerical variables α where C is an $\mathcal{ALCN}^S$-concept.

Since $\mathcal{ALCN}^S$ allows for full negation of concepts, universal quantification of numerical variables can be expressed: In the following, we us $(\uparrow\alpha.C)$ as a shorthand for $\neg(\downarrow\alpha.\neg C)$.

Before giving the semantics of $\mathcal{ALCN}^S$-concepts, we have to define what it means that a numerical variable occurs free in a concept:

Definition 6.3 The occurrence of a variable $\alpha \in N_V$ is said to be *bound in* C iff α occurs in the scope C' of a quantified subterm $(\downarrow\alpha.C')$ of C. Otherwise, the occurrence is said to be *free*. The set $\mathsf{free}(C) \subseteq N_V$ denotes the set of variables that occur free in C. A concept C is *closed* iff $\mathsf{free}(C) = \emptyset$. The concept $C[\alpha/n]$ is obtained from a concept C by substituting all free occurrences of α by n.

Note that, as usual, a variable can occur both free and bound in a concept, as, for example, α in $((= \alpha\ R) \sqcap (\downarrow\alpha.(\exists R.(> \alpha\ R))))$.

Using this notation, we can define the semantics of $\mathcal{ALCN}^S$-concepts.

Definition 6.4 An interpretation of a closed $\mathcal{ALCN}^S$-concept is an interpretation of an $\mathcal{ALCN}$-concept which satisfies additionally

$$(\downarrow\alpha.C)^{\mathcal{I}} = \bigcup_{n \in \mathbb{N}} (C[\alpha/n])^{\mathcal{I}}. \tag{6.1}$$

If C is not closed and $\mathsf{free}(C) = \{\alpha_1, \ldots, \alpha_n\}$ for $n \geq 1$ then

$$C^{\mathcal{I}} := (\downarrow\alpha_1.\ldots\downarrow\alpha_n.C)^{\mathcal{I}}.$$

This definition reduces symbolic number restrictions to traditional ones, hence their semantics is well-defined. Since $(\uparrow\alpha.C)$ is an abbreviation for $\neg(\downarrow\alpha.\neg C)$, we can give its semantics directly by

$$(\uparrow\alpha.C)^{\mathcal{I}} = \bigcap_{n \in \mathbb{N}} (C[\alpha/n])^{\mathcal{I}}.$$

Similar to $\mathcal{ALCN}$, it can be shown that $\mathcal{ALCN}^S$ still has the tree-model property. This can be proved by *unravelling* an arbitrary model $\mathcal{I}$ of C to a tree-model $\mathcal{I}'$ of C. This construction is similar to the one presented in [Thomas1992]. Let $x_0 \in C^{\mathcal{I}}$ be an instance of C in $\mathcal{I}$. Then $\mathcal{I}'$ is obtained by unwinding (cyclic) paths from x_0 to (infinite) paths, that is each path $< x_0 R_1 x_1 R_1 \ldots R_n x_n >$ in $\mathcal{I}$ corresponds to an individual in $\mathcal{I}'$ and vice versa. More precisely,

$$\begin{array}{rcl} \Delta^{\mathcal{I}'} & := & \{< x_0 R_1 x_1 R_1 \ldots R_n x_n >| \, (x_i, x_{i+1}) \in R^{\mathcal{I}}_{i+1} \text{ for } 0 \leq i \leq n-1\}, \\ A^{\mathcal{I}'} & := & \{< x_0 R_1 x_1 R_1 \ldots R_n x_n > \in \Delta^{\mathcal{I}'} \mid x_n \in A^{\mathcal{I}}\} \text{ for concept names } A, \\ R^{\mathcal{I}} & := & \{(< x_0 R_1 x_1 R_1 \ldots R_n x_n >, < x_0 R_1 x_1 R_1 \ldots R_n x_n R y >) \in \Delta^{\mathcal{I}'} \times \Delta^{\mathcal{I}'}\}. \end{array}$$

By induction on the structure of concept, it can easily be shown that $\mathcal{I}'$ is a tree-model of C. However, the resulting tree can be of infinite outdegree. In contrast to $\mathcal{ALCN}$, $\mathcal{ALCN}^S$ does not have the finite-model property. For example, the concept

$$(\uparrow\alpha.(\geq \alpha\ R)) \tag{6.2}$$

is satisfiable, but each instance of (6.2) has infinitely many R-successors. On the one hand, the interpretation $\mathcal{I}$ where

$$\begin{array}{rcl} \Delta^{\mathcal{I}} & := & \{x, y_0, y_1, y_2, \ldots\} \\ R^{\mathcal{I}} & := & \{(x, y_i) \mid i \in \mathbb{N}\} \end{array}$$

is clearly a model of (6.2). On the other hand, each model of (6.2) satisfies $\bigcap_{n \in \mathbb{N}} (\geq n\ R)^{\mathcal{I}} \neq \emptyset$, hence in $\Delta^{\mathcal{I}}$ there are infinitely many R-successors of some x.

This example shows that $\mathcal{ALCN}^S$ can enforce infinite models, which, intuitively, makes reasoning more complex. However, the attempt to construct a satisfiable concept that has only infinite models and that involves only existential quantification of numerical variables will fail—such a concept can only be constructed using universal quantification of numerical variables. Hence we introduce $\mathcal{ALUEN}^S$, a sublanguage of $\mathcal{ALCN}^S$ which is obtained by allowing only for *existential* quantification of numerical variables:

Definition 6.5 $\mathcal{ALUEN}^S$-concepts are those $\mathcal{ALCN}^S$-concepts where negation occurs only in front of concept names and number restrictions.

Since in $\mathcal{ALCN}^S$ universal quantification of numerical variables came only in as an abbreviation of negated existential quantification, all numerical variables in

$\mathcal{ALUEN}^S$ are therefore quantified existentially. Nevertheless, it is still a super-language of $\mathcal{ALCN}$. Furthermore, all examples given in Section 1.2 to motivate the introduction of symbolic number restrictions are $\mathcal{ALUEN}^S$-concepts. The reason for introducing first $\mathcal{ALCN}^S$ is that it is propositionally closed—which enables the reduction of subsumption to satisfiability and makes the syntax of $\mathcal{ALCN}^S$ easier to use. The reason for restricting it to $\mathcal{ALUEN}^S$ is that $\mathcal{ALCN}^S$ can be shown to have undecidable inference problems—whereas satisfiability of $\mathcal{ALUEN}^S$-concepts will be shown to be satisfiable.

6.1 Number Restrictions on Complex Roles

The examples given in the beginning of this chapter already highlighted the impact complex roles inside number restrictions have on the expressive power of $\mathcal{ALC}$. For a deeper insight into the expressiveness of these extensions, we first give the undecidability results: The concepts used in the proofs reveal the amount of expressive power added to $\mathcal{ALCN}$ by complex roles in number restrictions.

6.1.1 Undecidability results

In this section, undecidability of $\mathcal{ALC}_+\mathcal{N}(\circ, \sqcup)$, $\mathcal{ALCN}(\circ, \sqcup, ^{-1})$, $\mathcal{ALCN}(\circ, \sqcap)$, and $\mathcal{ALC}_+\mathcal{N}(\circ)$ will be shown by a reduction of the *domino problem*. This well-known, undecidable problem [Wang1963; Berger1966; Knuth1968] is used for the proof of all undecidability results in this thesis, and this section starts with its introduction.

The problem problem asks for the tiling of the (infinite) plane using a finite set of domino types; see Figure 6.1. Intuitively, each domino is a square with coloured edges. The tiling may not have any holes, edges of neighbouring dominos must have the same colour on the touching edges, and dominos may not be turned or flipped.

Definition 6.6 A *tiling system* $\mathcal{D} = (D, H, V)$ is given by a non-empty set $D = \{D_1, \dots, D_\ell\}$ of *domino types*, and by horizontal and vertical *matching pairs* $H \subseteq D \times D$, $V \subseteq D \times D$. The *domino problem* asks for a *compatible tiling* of the first quadrant $\mathbb{N} \times \mathbb{N}$ of the plane, i.e., a mapping $t : \mathbb{N} \times \mathbb{N} \to D$ such that for all $m, n \in \mathbb{N}$:

$$(t(m,n), t(m+1,n)) \in H \text{ and } (t(m,n), t(m,n+1)) \in V.$$

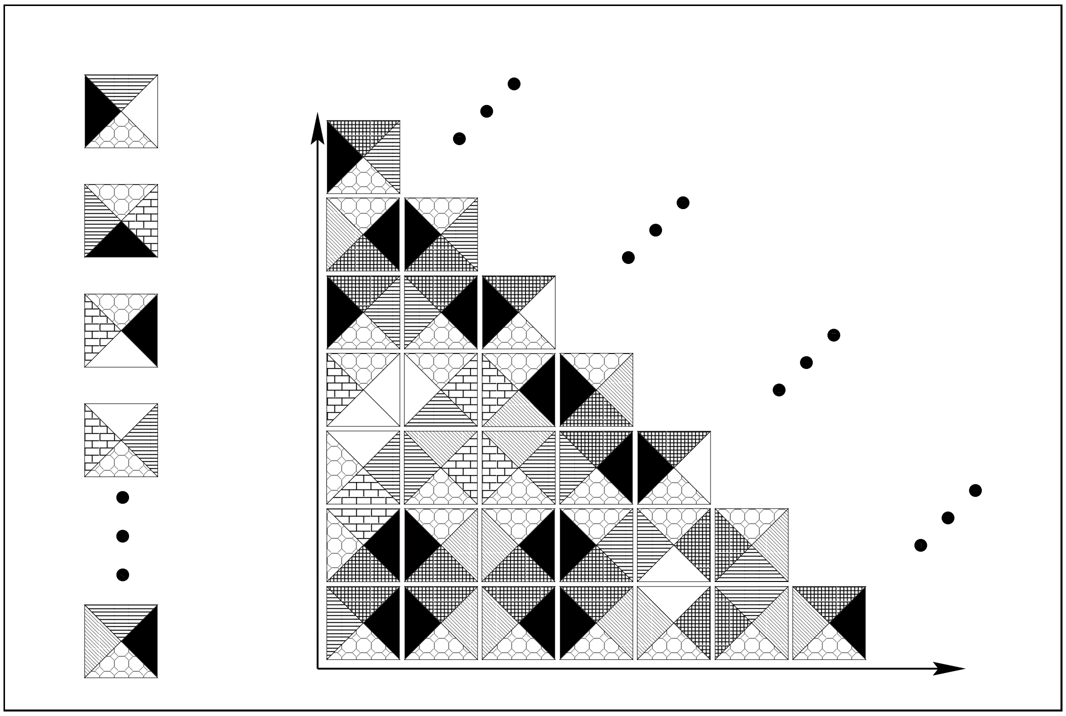

Figure 6.1: A set of domino types and a first part of a tiling.

The standard domino problem asks for a compatible tiling of the whole plane. However, a compatible tiling of the first quadrant yields compatible tilings of arbitrarily large finite rectangles, which in turn yield a compatible tiling of the plane [Knuth1968]. Thus, the undecidability result for the standard problem [Berger1966] carries over to this variant.

In order to reduce the domino problem to satisfiability of concepts, we must show how a given tiling system $\mathcal{D}$ can be translated into a concept $E_{\mathcal{D}}$ (of the language under consideration) such that $E_{\mathcal{D}}$ is satisfiable iff $\mathcal{D}$ allows for a compatible tiling. This task can be split into three subtasks, which will be first explained on an intuitive level, before showing how they can be achieved for the four Description Logics under consideration.

Task 1: It must be possible to represent a single "square" of $\mathbb{N} \times \mathbb{N}$, which consists of points (n, m), $(n, m+1)$, $(n+1, m)$, and $(n+1, m+1)$. The idea is to introduce roles X, Y, where X goes one step into the horizontal (i.e. x-) direction, and Y goes one step into the vertical (i.e. y-) direction. The concept language must be expressive enough to describe that an individual (a point (n, m)) has exactly one X-successor (the point $(n +$

$1, m)$), exactly one Y-successor (the point $(n, m+1)$), and that the $X \circ Y$-successor coincides with the $Y \circ X$-successor (the point $(n+1, m+1)$).

Task 2: It must be possible to express that a tiling is locally compatible, i.e., that the X- and Y-successors of a point have an admissible domino type. The idea is to associate each domino type D_i with an atomic concept D_i, and to express the horizontal and vertical matching conditions via value restrictions on the roles X, Y.

Task 3: It must be possible to impose the above *local* conditions on all points in $\mathbb{N} \times \mathbb{N}$. This can be achieved by constructing a "universal" role U and a "start" individual such that every point is a U-successor of this start individual. The local conditions can then be imposed on all points via value restrictions on U for the start individual.

Task 2 is rather easy, and can be realized using the $\mathcal{ALC}$-concept $C_\mathcal{D}$ given in Figure 6.2. The first conjunct expresses that every point has exactly one domino type, and the value restrictions in the second conjunct express the horizontal and vertical matching conditions. Disjunction occurs in both conjuncts: In the first, disjunction is used to express that each point is associated to a domino type D_1 *or* a domino type D_2 etc. In the second conjunct, disjunction is used to express the compatibility condition, which is of the form "*if* a point is associated to a domino D_i, *then* its X-successor (resp. its Y-successor) must be associated to a domino D_j such that $(D_i, D_j) \in H$ (resp. $\in V$)". In contrast to this, Task 1 and Task 3 can be achieved without using disjunction of concepts.

Task 1 can be achieved in any extension of $\mathcal{ALCN}(\circ)$ with either union or intersection of roles in number restrictions: see the concepts $C_\sqcap$ and $C_\sqcup$ in Figure 6.2.

Task 3 is easy for languages that extend $\mathcal{ALC}_{\mathsf{reg}}$, and more difficult for languages without at least the transitive closure. The general idea is that the start individual s is an instance of the concept $E_\mathcal{D}$ to be constructed. From this individual, one can reach via U the origin $(0,0)$ of $\mathbb{N} \times \mathbb{N}$, and each point that is connected with the origin via some arbitrary X- and Y-path.

With the image of these tasks in mind, the reduction concepts are now explained in detail for each undecidable extension of $\mathcal{ALCN}$, $\mathcal{ALC}_{\mathsf{reg}}\mathcal{N}$, and $\mathcal{ALC}_+\mathcal{N}$ by complex number restrictions.

$$
\begin{array}{ll}
C_{\mathcal{D}} := & \bigsqcup_{1\leq i\leq m} (D_i \sqcap (\bigsqcap_{1\leq j\leq m, i\neq j} \neg D_j)) \sqcap \\
& \bigsqcap_{1\leq i\leq m} (D_i \Rightarrow ((\forall X.(\bigsqcup_{(D_i,D_j)\in H} D_j)) \sqcap (\forall Y.(\bigsqcup_{(D_i,D_j)\in V} D_j)))) \\
C_{\sqcup} := & (=1\ X) \sqcap (=1\ Y) \sqcap (=1\ X\circ Y) \sqcap (=1\ Y\circ X) \sqcap \\
& (=1\ Y\circ X \sqcup X\circ Y) \\
C_{\sqcap} := & (=1\ X) \sqcap (=1\ Y) \sqcap (=1\ X\circ Y) \sqcap (=1\ Y\circ X) \sqcap \\
& (=1\ Y\circ X \sqcap X\circ Y)
\end{array}
$$

$$E_{\mathcal{D}}^{(1')} := (=1\ R) \sqcap (\forall R^+.(C_{\sqcup} \sqcap C_{\mathcal{D}} \sqcap (\geq 2\ R) \sqcap (\leq 2\ R \sqcup X \sqcup Y)))$$

$$
\begin{array}{ll}
E_{\mathcal{D}}^{(2)} := & (\geq 1\ U) \sqcap (\forall U.\ (C_{\sqcup} \sqcap C_{\mathcal{D}} \sqcap (=1\ X\circ U^{-1}) \sqcap (=1\ Y\circ U^{-1}) \sqcap \\
& \qquad (\leq 1\ U^{-1} \sqcup Y\circ U^{-1} \sqcup X\circ U^{-1}))) \\
E_{\mathcal{D}}^{(3)} := & (\geq 1\ R \sqcap R\circ T\circ R) \sqcap \\
& (\forall R.\forall T.\forall R.\ (C_{\sqcap} \sqcap C_{\mathcal{D}} \sqcap (\leq 1\ T) \sqcap \\
& \qquad (\forall Y.(\leq 1\ T)) \sqcap (\forall X.(\leq 1\ T)) \sqcap \\
& \qquad (=1\ T \sqcap X\circ T \sqcap Y\circ T) \sqcap \\
& \qquad (=1\ X \sqcap X\circ T\circ R) \sqcap (=1\ Y \sqcap Y\circ T\circ R)))
\end{array}
$$

where $A \Rightarrow B$ is an abbreviation for $\neg A \sqcup B$ and
$(= n\ R)$ is an abbreviation for $(\geq n\ R) \sqcap (\leq n\ R)$.

Figure 6.2: Concepts used in the proof of Theorem 6.7.

(1) We start with $\mathcal{ALC}_{\mathsf{reg}}\mathcal{N}$ since here it is rather easy to reach all individuals representing points in the plane from the start individual. In extensions of $\mathcal{ALC}_{\mathsf{reg}}\mathcal{N}$, we can use the complex role $(X \sqcup Y)^+$ to reach every point. Thus, for each tiling system $\mathcal{D}$ the $\mathcal{ALC}_{\mathsf{reg}}\mathcal{N}(\circ)$-concept

$$E_{\mathcal{D}}^{(1)} := C_{\sqcup} \sqcap C_{\mathcal{D}} \sqcap \forall (X \sqcup Y)^+.(C_{\sqcup} \sqcap C_{\mathcal{D}}).$$

can be constructed which is obviously satisfiable if and only if $\mathcal{D}$ admits a compatible tiling.

The complex role in the value restriction can even be restricted to a simple transitive closure of an atomic role. Intuitively, a starting point outside the plane is used which is connected to each point in the plane via some R-path. To achieve this, the concept $E_{\mathcal{D}}^{(1')}$ in Figure 6.2 makes sure that the X- and the Y-successors of each point in the plane are also R-successors of this point. Hence R^+ can be used in place of $(X \sqcup Y)^+$ as "universal" role and thus the concept $E_{\mathcal{D}}^{(1')}$ is in $\mathcal{ALC}_+\mathcal{N}(\circ, \sqcup)$.

(2) In $\mathcal{ALCN}(\circ, \sqcup, ^{-1})$, a role name U for the "universal" role is explicitly introduced, and number restrictions which involve composition, union, and inversion of roles are used to make sure that the start individual is directly connected via U with every point: see the concept $E_{\mathcal{D}}^{(2)}$ in Figure 6.2. The number restrictions inside the value restriction make sure that every point p that is reached via U from the start individual satisfies the following: Its X-successor and its Y-successor each have exactly one U-predecessor, which coincides with the (unique) U-predecessor of p, i.e., the start individual. Thus, the X-successor and the Y-successor of p are also U-successors of the start individual.

(3) For $\mathcal{ALCN}(\circ, \sqcap)$, a similar construction is possible: Since inversion of roles is not allowed in $\mathcal{ALCN}(\circ, \sqcap)$, two role names R and T are needed for the construction of the universal role. The intuition is that T plays the rôle of the inverse of R (except for one individual), and the "universal" role corresponds to the composition $R \circ T \circ R$: The start individual s (which is an instance of $E_{\mathcal{D}}^{(3)}$), has at least one R-successor $p_{(0,0)}$, which coincides with its $R \circ T \circ R$-successor. The individual $p_{(0,0)}$ corresponds to the origin of $\mathbb{N} \times \mathbb{N}$. Let s' be the $R \circ T$-successor of s. The number restrictions of $E_{\mathcal{D}}^{(3)}$ make sure that $p_{(0,0)}$ satisfies the following: It has exactly one T-successor, namely s', which coincides with the (unique) T-successors of its X- and Y-successors. In addition, the (unique) X-successor of $p_{(0,0)}$ is also an $X \circ T \circ R$-successor of $p_{(0,0)}$, which makes sure that the X-successor of $p_{(0,0)}$ is an R-successor of s', and thus an $R \circ T \circ R$-successor of s. The same holds for the Y-successor. One can now continue the argument with the X-successor (resp. Y-successor) of $p_{(0,0)}$ in place of $p_{(0,0)}$.

With the intuition given above, it is not hard to show for all $i, 1 \leq i \leq 3$, that a tiling system $\mathcal{D}$ has a compatible tiling iff $E_{\mathcal{D}}^{(i)}$ is satisfiable.

Theorem 6.7 Satisfiability (and thus also subsumption) of concepts is undecidable for $\mathcal{ALC}_{+}\mathcal{N}(\circ, \sqcup)$, $\mathcal{ALCN}(\circ, \sqcup, ^{-1})$, and $\mathcal{ALCN}(\circ, \sqcap)$.

Proof: Since the other two cases are similar and easier, it will only be shown that the $\mathcal{ALCN}(\circ, \sqcap)$-concept $E_{\mathcal{D}}^{(3)}$ is satisfiable if, and only if, $\mathcal{D}$ admits a compatible tiling.

The "if" direction: Let $\mathcal{D}$ be a tiling system, $E_{\mathcal{D}}^{(3)}$ the corresponding concept, and t a compatible tiling. Then we define a model $\mathcal{I}$ of $E_{\mathcal{D}}^{(3)}$ as follows:

$$
\begin{array}{rcl}
\Delta^{\mathcal{I}} & := & \{s\} \cup \{p_{(n,m)} \mid n,m \in \mathbb{N}\} \\
X^{\mathcal{I}} & := & \{(p_{(n,m)}, p_{(n+1,m)}) \mid n,m \in \mathbb{N}\} \\
Y^{\mathcal{I}} & := & \{(p_{(n,m)}, p_{(n,m+1)}) \mid n,m \in \mathbb{N}\} \\
R^{\mathcal{I}} & := & \{(s, p_{(n,m)}) \mid n,m \in \mathbb{N}\} \\
T^{\mathcal{I}} & := & \{(p_{(n,m)}, s) \mid n,m \in \mathbb{N}\} \\
D_i^{\mathcal{I}} & := & \{p_{(n,m)} \mid t(n,m) = D_i\} \text{ for each } D_i \in D
\end{array}
$$

Since each $p_{(n,m)}$ is both an R- and an $R \circ T \circ R$-successor of s, s obviously is an instance of the first conjunct of $E_{\mathcal{D}}^{(3)}$. Furthermore, the value restriction $\forall R.\forall T.\forall R$ applies to each $p_{(n,m)}$. Now, since t is compatible, each individual $p_{(n,m)}$ is an element of exactly one $D_i^{\mathcal{I}}$, and both the vertical and horizontal matching conditions are satisfied. Hence each $p_{(n,m)}$ is an instance of $C_{\mathcal{D}}$. Furthermore, X, Y are defined in such a way that each $p_{(n,m)}$ has exactly one X-successor $p_{(n+1,m)}$, one Y-successor $p_{(n,m+1)}$, and one $X \circ Y$-successor $p_{(n+1,m+1)}$ which coincides with its $Y \circ X$-successor. Hence each $p_{(n,m)}$ is an element of $C_{\sqcap}^{\mathcal{I}}$.

Now, since s is the only T-successor, each $p_{(n,m)}$ is clearly an instance of

$$(\leq 1\ T) \sqcap (\forall Y.(\leq 1\ T)) \sqcap (\forall X.(\leq 1\ T)).$$

Furthermore, s is the common T-successor of all $p_{(n,m)}$, hence each $p_{(n,m)}$ is an instance of $(= 1\ T \sqcap X \circ T \sqcap Y \circ T)$. Finally, since each $p_{(n,m)}$ has exactly one X-successor $p_{(n+1,m)}$ and one Y-successor $p_{(n,m+1)}$, and since $p_{(n+1,m)}$ and $p_{(n,m+1)}$ are their own $T \circ R$-successors, each $p_{(n,m)}$ is clearly an instance of

$$(= 1\ X \sqcap X \circ T \circ R) \sqcap (= 1\ Y \sqcap Y \circ T \circ R).$$

The "only if" direction: Let $\mathcal{I}$ be some model of $E_{\mathcal{D}}^{(3)}$ with $s \in E_{\mathcal{D}}^{(3)}$, let $p_{(0,0)}$ be some R-successor of r which is also an $R \circ T \circ R$-successor s (because of the first conjunct of $E_{\mathcal{D}}^{(3)}$, such a $p_{(0,0)}$ exists), and let s' be the $R \circ T$-successor of s (which possibly coincides with s). Since each $R \circ T \circ R$-successor of s is an instance of $C_{\mathcal{D}}$, $p_{(0,0)}$ is an instance of exactly one $D_j \in D$. Define a compatible tiling t by

$$
\begin{array}{rcl}
t(0,0) & := & D_j \\
t(n,m) & := & D_i \text{ if } p \text{ is an } X^n \circ Y^m\text{-successor of } p_{(0,0)} \text{ and } p \in D_i^{\mathcal{I}}
\end{array}
$$

It remains to be shown that (1) t is well-defined and (2) the horizontal and the vertical matching conditions are satisfied.

(1) t is well-defined because (a) $p_{(0,0)}$ has, for each $n, m \in \mathbb{N}$, exactly one $X^n \circ Y^m$-successor which (b) is furthermore instance of exactly one D_i. (a) holds because the value restriction $\forall R.\forall T.\forall R.\ldots$ applies to $p_{(0,0)}$, hence $p_{(0,0)}$ is an instance of $C_\sqcap$, which means that it has exactly one X-successor, one Y-successor, and one $X \circ Y$-successor, which coincides with its unique $Y \circ X$-successor. Furthermore, $p_{(0,0)}$ is an instance of

$$(\leq 1\ T) \sqcap (\forall Y.(\leq 1\ T)) \sqcap (\forall X.(\leq 1\ T)) \sqcap$$
$$(= 1\ T \sqcap X \circ T \sqcap Y \circ T) \sqcap (= 1\ X \sqcap X \circ T \circ R) \sqcap (= 1\ Y \sqcap Y \circ T \circ R).$$

Hence s' is also a T-successor of the X- and Y-successors $p_{(1,0)}, p_{(0,1)}$ of $p_{(0,0)}$, and $p_{(1,0)}, p_{(0,1)}$ are both R-successors of s'. Therefore $p_{(1,0)}, p_{(0,1)}$ are both $R \circ T \circ R$-successors of s, and thus the value restriction $\forall R.\forall T.\forall R.\ldots$ applies also to $p_{(1,0)}, p_{(0,1)}$. The same arguments as for $p_{(0,0)}$ can then be used on $p_{(1,0)}, p_{(0,1)}$, hence for each n, m, there is exactly one $X^n \circ Y^m$-successor of $p_{(0,0)}$. (b) is due to fact that $p_{(0,0)}$ as well as all its $X^n \circ Y^m$-successors are an instance of $C_{\mathcal{D}}$.

(2) Obviously, $C_{\mathcal{D}}$ ensures that the horizontal and the vertical matching conditions are satisfied. ■

In the overview given in Figure 6.3, decidable extensions are hatched horizontally, whereas undecidable ones are hatched vertically. Given

- the above results,
- the fact that, as a consequence of a result in [Grädel *et al.*1997], $\mathcal{ALCN}(^{-1}, \sqcap, \sqcup)$ is decidable, and
- the fact that $\mathcal{ALCN}(\circ)$ is decidable (which will be proved in the next section),

the only problems which remain open for the extensions of $\mathcal{ALCN}$ concern $\mathcal{ALCN}(\circ, ^{-1})$ and $\mathcal{ALCN}(\circ, \sqcup)$. Unfortunately, these problems must remain open for the time being: neither a decision procedure for one of these extensions nor a proof of their undecidability could be found.

Extensions of $\mathcal{ALC}_+\mathcal{N}$

So far, (un)decidability of two extensions of $\mathcal{ALC}_+\mathcal{N}$ by complex roles was proved: In [De Giacomo1995], decidability of $\mathcal{ALC}_+\mathcal{N}(^{-1})$ is shown and Theorem 6.7 is concerned with the undecidability of $\mathcal{ALC}_+\mathcal{N}(\circ, \sqcup)$. It will now be

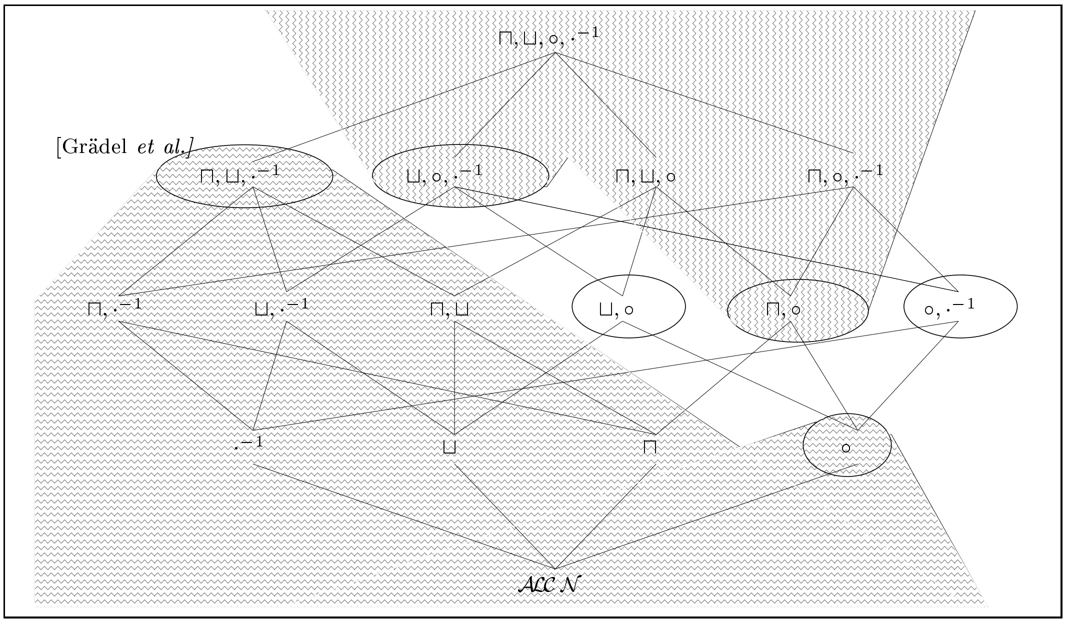

Figure 6.3: (Un)decidability results for extension of $\mathcal{ALCN}$.

shown that, in $\mathcal{ALC}_{+}\mathcal{N}$, it suffices to allow for composition in number restrictions in order to lose decidability. In Figure 6.4, an overview of these results is given, again with decidable extensions hatched horizontally and undecidable ones vertically.

Again, a reduction of the domino problem to concept satisfiability is used to show undecidability of $\mathcal{ALC}_{+}\mathcal{N}(\circ)$. Since this reduction is rather different from the ones above and more complicated, it is treated separately. The (redundant) reduction for $\mathcal{ALC}_{+}\mathcal{N}(\circ, \sqcup)$ was given since it served to give the intuition for $\mathcal{ALCN}(\circ, \sqcup, ^{-1})$ and $\mathcal{ALCN}(\circ, \sqcap)$. The concepts used for the reduction of the domino problem to $\mathcal{ALC}_{+}\mathcal{N}(\circ)$-concept satisfiability are given in Figure 6.6.

As neither $C_{\sqcap}$ nor $C_{\sqcup}$ is an $\mathcal{ALC}_{+}\mathcal{N}(\circ)$-concept, Task 1 must be accomplished differently. Instead of using a "horizontal" role X and a "vertical" role Y, only a single role X can be used—otherwise, one could not express that the horizontal-vertical successor coincides with the vertical-horizontal successor of a point.

The three tasks introduced in the beginning of Section 6.1.1 are then accomplished as follows.

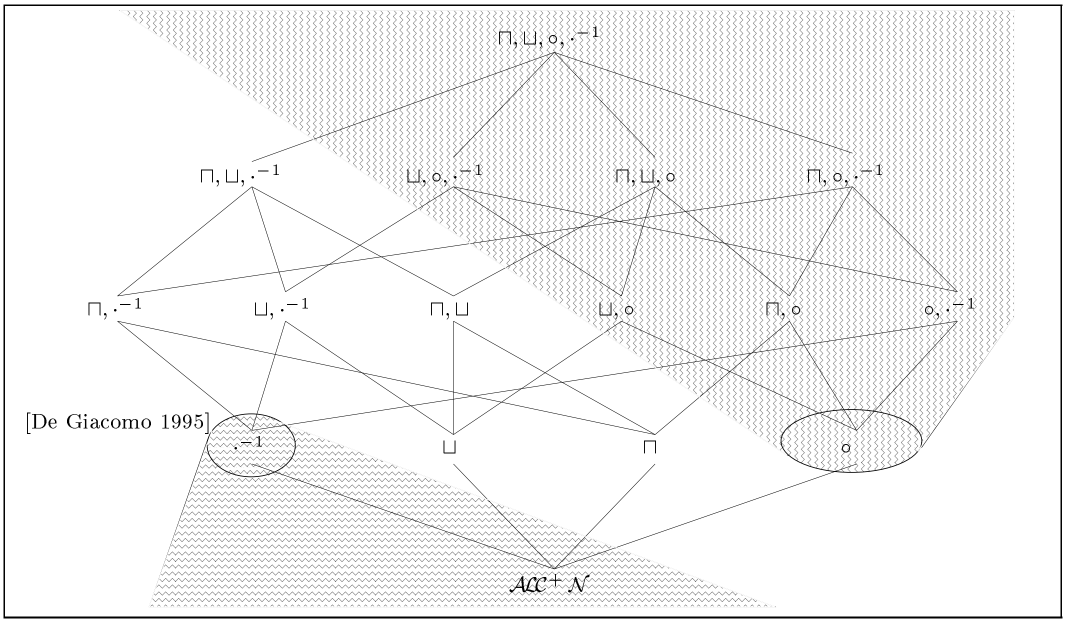

Figure 6.4: (Un)decidability results for extension of $\mathcal{ALC}_+\mathcal{N}$.

Task 1 The concept $C_\square$ describes a square by using a single role X. Each instance of $C_\square$ has two X-successors that in turn each have two X-successors. The conjunct $(= 3\ X \circ X)$ makes sure that the X-successors of an instance of $C_\square$ have one common X-successor.

Task 3 is easy because $\mathcal{ALC}_+\mathcal{N}(\circ)$ allows for the transitive closure of roles. If $s \in {E_\mathcal{D}^{(4)}}^\mathcal{I}$, then s has exactly one X-successor, say $p_{(0,0)}$, which is an instance of A. Each point in the grid is an X^n-successor of s. Thus the local conditions on all points in the grid are imposed by $\forall X^+(C_\square \sqcap C_{\text{prim}} \sqcap C_{\text{diag}} \sqcap C_\mathcal{D})$.

Task 2 is difficult because we must distinguish between the "horizontal" and the "vertical" X-successor of a point. For this purpose, the concepts A, B, and C are used in the following way (see Figure 6.5 for a better intuition): The concept C_{prim} enforces that each point is an instance of either A or B or C, and that exactly one domino type D_i is associated with each point. The concept C_{diag} makes sure that each instance of A has one X-successor in B and one in C, and similar for instances of B and C. Without loss of generality, when visualising the grid in Figure 6.5, we have drawn the X-successor of $p_{0,0}$ which is in C to its right and called it $p_{1,0}$. The other X-successor of $p_{0,0}$,

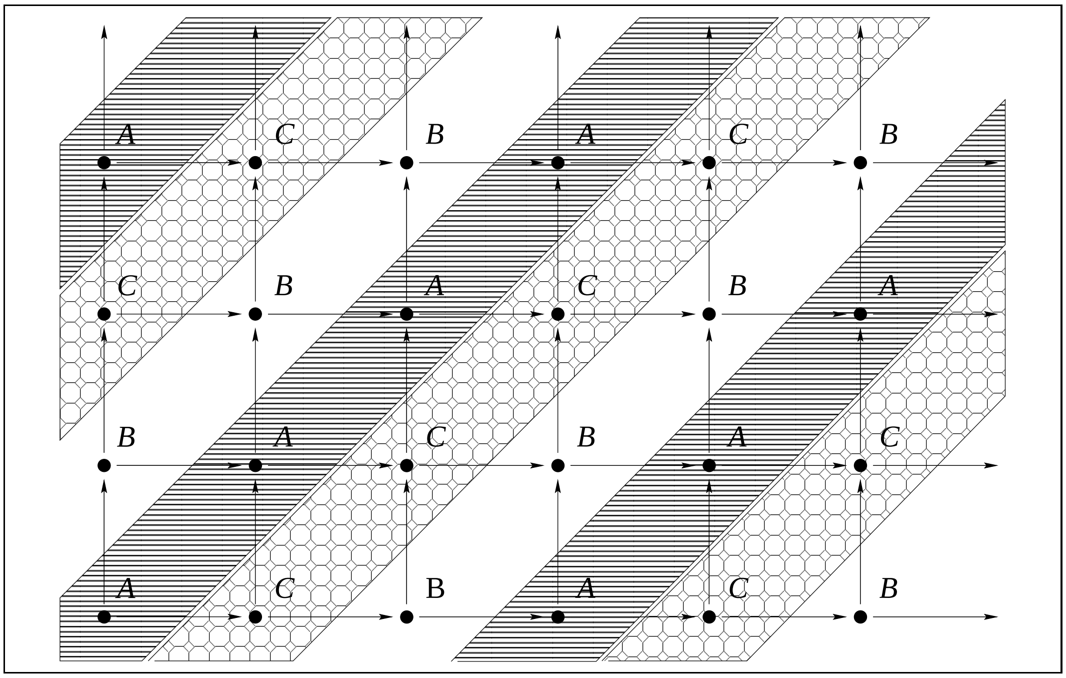

Figure 6.5: Visualisation of the grid as enforced by the $\mathcal{ALC}_+\mathcal{N}(\circ)$ reduction concept.

which is in B, is called $p_{0,1}$ and is drawn above it. Then it is easy to see that the remainder of the grid is determined in the sense that

- for each diagonal in the grid there is an $E \in \{A, B, C\}$ such that all points on this diagonal are instances of E,
- horizontal successors of points in A are always in C, of points in C are always in B, and of points in B are always in A,
- vertical successors of points in A are always in B, of points in B are always in C, and of points in C are always in A.

With these observations, $C_\mathcal{D}$ expresses the horizontal and vertical matching conditions.

With the intuition given above, it is not hard to show that a tiling system $\mathcal{D}$ has a compatible tiling iff $E_\mathcal{D}^{(4)}$ is satisfiable.[1]

[1] This concept is, to make it easier to understand, longer than necessary. The subconcept C_{diag} can be omitted because it is subsumed by $C_\mathcal{D} \sqcap C_{\text{prim}}$.

$$
\begin{array}{lll}
C_{\Box} & := & (= 2\ X) \sqcap (\forall X.(= 2\ X)) \sqcap (= 3\ X \circ X) \\
C_{\text{prim}} & := & (A \sqcup B \sqcup C) \sqcap \bigsqcup_{1 \le i \le m} (D_i \sqcap (\bigsqcap_{1 \le j \le m_{\bar{i}} \ne j} \neg D_j)) \\
C_{\text{diag}} & := & (A \Rightarrow ((\exists X.B) \sqcap (\exists X.C))) \sqcap \\
 & & (B \Rightarrow ((\exists X.A) \sqcap (\exists X.C))) \sqcap \\
 & & (C \Rightarrow ((\exists X.A) \sqcap (\exists X.B))) \\
C_{\mathcal{D}} & := & \bigsqcap_{1 \le i \le m} \Big(((A \sqcap D_i) \Rightarrow (\exists X.(C \sqcap (\bigsqcup_{(D_i,D_j) \in H} D_j)) \sqcap \\
 & & \qquad \exists X.(B \sqcap (\bigsqcup_{(D_i,D_j) \in V} D_j)))) \sqcap \\
 & & \quad ((B \sqcap D_i) \Rightarrow (\exists X.(A \sqcap (\bigsqcup_{(D_i,D_j) \in H} D_j)) \sqcap \\
 & & \qquad \exists X.(C \sqcap (\bigsqcup_{(D_i,D_j) \in V} D_j)))) \sqcap \\
 & & \quad ((C \sqcap D_i) \Rightarrow (\exists X.(B \sqcap (\bigsqcup_{(D_i,D_j) \in H} D_j)) \sqcap \\
 & & \qquad \exists X.(A \sqcap (\bigsqcup_{(D_i,D_j) \in V} D_j)))) \Big) \\
E^{(4)}_{\mathcal{D}} & := & (= 1\ X) \sqcap (\exists X.A) \sqcap (\forall X^{+}.(C_{\Box} \sqcap C_{\text{prim}} \sqcap C_{\text{diag}} \sqcap C_{\mathcal{D}}))
\end{array}
$$

where A, B and C are disjoint concepts. They are abbreviations for
$A := A_1, \qquad B := \neg A_1 \sqcap A_2 \qquad C := \neg A_1 \sqcap \neg A_2$

Figure 6.6: Concepts used in the proof of Theorem 6.8.

Theorem 6.8 Satisfiability (and thus also subsumption) of concepts is undecidable for $\mathcal{ALC}_{+}\mathcal{N}(\circ)$.

6.1.2 A decidable extension

In this section, a tableau-based algorithm for deciding satisfiability of $\mathcal{ALCN}(\circ)$-concepts is presented. The algorithm and the proof of its correctness are very similar to existing algorithms and proofs for languages with number restrictions on atomic roles [Hollunder *et al.*1990; Hollunder & Baader1991]. Please recall that the presence of number restrictions on role chains has as a consequence the loss of the tree-model property.

Nevertheless, the models generated by this algorithm are very similar to tree models. When started with a concept C_0, the algorithm generates a model of C_0 where every element can be reached from an initial root element, which is an instance of C_0, via role chains. Furthermore, the root does not have a role predecessor, and every role chain from the root to an element has the same length—even though there may exist more than one such chain. This fact will become important in the proof of termination.

As usual, without loss of generality all concepts are supposed to be in negation normal form (NNF); see Section 4.2. Similar to the algorithm presented in Figure 4.4, the new algorithm works on constraints, but since it has to handle number restrictions on role chains, we need an additional kind of constraints and an extended definition of the notion a clash:

Definition 6.9 Let $\tau = \{x, y, z, \dots\}$ be a countably infinite set of individual variables. A *constraint* is either of the form

xRy, where R is a role name in N_R and $x, y \in \tau$,

$x : D$ for some $\mathcal{ALCN}(\circ)$-concept D in NNF and some $x \in \tau$, or

$x \neq y$ for $x, y \in \tau$.[2]

As in Definition 4.6, a *constraint system* is a set of constraints, and $\tau_S \subseteq \tau$ denotes the individual variables occurring in a constraint system S.

An interpretation $\mathcal{I}$ is a *model of a constraint system* S iff there is a mapping $\pi : \tau_S \to \Delta^{\mathcal{I}}$ such that $\mathcal{I}, \pi$ satisfy each constraint in S, i.e.,

$$\begin{aligned} (\pi(x), \pi(y)) \in R^{\mathcal{I}} &\quad \text{for all } xRy \in S, \\ \pi(x) \in D^{\mathcal{I}} &\quad \text{for all } x : D \in S, \\ \pi(x) \neq \pi(y) &\quad \text{for all } (x \neq y) \in S. \end{aligned}$$

Similar to Definition 4.6, y is said to be an $R_1 \circ \ldots \circ R_m$-successors of x in S if there exist $y_1, \dots, y_m \in \tau_S$ with

$$\begin{aligned} & xR_1y_1 \in S, \\ & \text{for all } 2 \leq i \leq m,\ y_{i-1}R_iy_i \in S,\ \text{and} \\ & y_m = y. \end{aligned}$$

S contains a *clash* iff $\{x : A, x : \neg A\} \subseteq S$ for some concept name A and some variable $x \in \tau_S$, or $x : (\leq n\ R_1 \circ \ldots \circ R_m) \in S$ and x has $\ell > n$ $R_1 \circ \ldots \circ R_m$-successors $y_1, \dots, y_\ell$ in S such that for all $i \neq j$ we have $y_i \neq y_j \in S$. A constraint system S is called *complete* iff none of the completion rules given in Figure 6.7 can be applied to S. In these rules, the constraint system $S[y_2/y_1]$ is obtained from S by substituting each occurrence of y_2 in S by y_1.

[2]We consider such inequalities as being symmetric, i.e., if $x \neq y$ belongs to a constraint system, then $y \neq x$ (implicitly) belongs to it as well.

Figure 6.7 introduces the *completion rules* that are used to test $\mathcal{ALCN}(\circ)$-concepts for satisfiability. Similar to the algorithm presented in Figure 4.4, the *completion algorithm* works on a tree where each node is labelled with a constraint system. It starts with the tree consisting of a root labelled with $S = \{x_0 : C_0\}$, where C_0 is the $\mathcal{ALCN}(\circ)$-concept in NNF to be tested for satisfiability. A rule can only be applied to a leaf labelled with a clash-free constraint system. Applying a rule $S \to S_i$, for $1 \leq i \leq n$, to such a leaf leads to the creation of n new successors of this node, each labelled with one of the constraint systems S_i. The algorithm terminates if none of the rules can be applied to any of the leaves. In this situation, it answers with "C_0 is satisfiable" iff one of the leaves is labelled with a clash-free constraint system.

Soundness and completeness of this algorithm is an immediate consequence of the following facts:

Lemma 6.10 Let C_0 be an $\mathcal{ALCN}(\circ)$-concept in NNF, and let S be a constraint system obtained by applying the completion rules to $\{x_0 : C_0\}$. Then

1. For each completion rule $\mathcal{R}$ that can be applied to S, and for each interpretation $\mathcal{I}$, (i) and (ii) are equivalent.
   (i) $\mathcal{I}$ is a model of S.
   (ii) $\mathcal{I}$ is a model of one of the systems S_i obtained by applying $\mathcal{R}$.
2. If S is a complete and clash-free constraint system, then S has a model.
3. If S contains a clash, then S does not have a model.
4. The completion algorithm terminates when applied to $\{x_0 : C_0\}$.

Since Lemma 6.10 is the same as Lemma 4.8 (besides the different logics), the arguments which were used to show that Lemma 4.8 implies decidability of satisfiability of $\mathcal{ALC}$-concepts now yield decidability of $\mathcal{ALCN}(\circ)$:

Theorem 6.11 Subsumption and satisfiability of $\mathcal{ALCN}(\circ)$-concepts is decidable.

It remains to prove Lemma 6.10.

1. Conjunction: If $x:(C_1 \sqcap C_2) \in S$ and $x:C_1 \notin S$ or $x:C_2 \notin S$, then
$S \rightarrow \quad S \cup \{x:C_1, x:C_2\}$

2. Disjunction: If $x:(C_1 \sqcup C_2) \in S$ and $x:C_1 \notin S$ and $x:C_2 \notin S$, then
$S \rightarrow \quad S_1 = S \cup \{x:C_1\}$
$S \rightarrow \quad S_2 = S \cup \{x:C_2\}$

3. Value restriction: If $x:(\forall R.C) \in S$ for a role name R, y is an R-successor of x in S and $y:C \notin S$, then
$S \rightarrow \quad S \cup \{y:C\}$

4. Existential restriction: If $x:(\exists R.C) \in S$ for a role name R and there is no R-successor y of x in S with $y : C \in S$, then
$S \rightarrow \quad S \cup \{xRz, z : C\}$ for a new variable $z \in \tau \setminus \tau_S$.

5. Number restriction: If $x:(\geq n\ R_1 \circ \ldots \circ R_m) \in S$ for role names $R_1, \ldots, R_m$ and x has less than n $R_1 \circ \ldots \circ R_m$-successors in S, then
$S \rightarrow \quad S \cup \{xR_1y_2, y_mR_mz\} \cup \{y_iR_iy_{i+1} \mid 2 \leq i \leq m-1\} \cup$
$\qquad \{z \neq w \mid w \text{ is an } R_1 \circ \ldots \circ R_m\text{-successor of } x \text{ in } S\}$
where z, y_i are new variables in $\tau \setminus \tau_S$.

6. Number restriction: If $x:(\leq n\ R_1 \circ \ldots \circ R_m) \in S$, x has more than n $R_1 \circ \ldots \circ R_m$-successors in S, and there are $R_1 \circ \ldots \circ R_m$-successors y_1, y_2 of x in S with $(y_1 \neq y_2) \notin S$, then
$S \rightarrow S_{y_1,y_2} = S[y_2/y_1]$
for all pairs y_1, y_2 of $R_1 \circ \ldots \circ R_m$-successors of x with $(y_1 \neq y_2) \notin S$.

Figure 6.7: The completion rules for $\mathcal{ALCN}(\circ)$.

Proof of Part 1 of Lemma 6.10: We consider only the rules concerned with number restrictions, since the proof for Rules 1–4 is just as for $\mathcal{ALC}$.

5. Number restriction: Assume that the rule is applied to the constraint $x:(\geq n\ R_1 \circ \ldots \circ R_m)$, and that its application yields

$$S' = S \cup \{xR_1y_2, y_mR_mz\} \cup \{y_iR_iy_{i+1} \mid 2 \leq i \leq m-1\} \\ \cup \{z \neq w \mid w \text{ is an } R_1 \circ \ldots \circ R_m\text{-successor of } x \text{ in } S\}.$$

Since S is a subset of S', any model of S' is also a model of S. Conversely, assume that $\mathcal{I}$ is a model of S, and let $\pi : \tau_S \rightarrow \Delta^{\mathcal{I}}$ be the corresponding mapping of individual variables to elements of $\Delta^{\mathcal{I}}$. On the one hand, since $\mathcal{I}$ satisfies $x:(\geq n\ R_1\circ\ldots\circ R_m)$, $\pi(x)$ has at least n $R_1\circ\ldots\circ R_m$-successors in $\mathcal{I}$. On the other hand, since Rule 5 is applicable to $x:(\geq n\ R_1\circ\ldots\circ R_m)$, x has less than n $R_1\circ\ldots\circ R_m$-successors in S. Thus, there exists an $R_1\circ\ldots\circ R_m$-successor b of $\pi(x)$ in $\mathcal{I}$ such that $b\neq\pi(w)$ for all $R_1\circ\ldots\circ R_m$-successors w of x in S. Let $b_2,\ldots,b_m\in\Delta^{\mathcal{I}}$ be such that $(\pi(x),b_2)\in R_1^{\mathcal{I}},(b_2,b_3)\in R_2^{\mathcal{I}},\ldots,(b_m,b)\in R_m^{\mathcal{I}}$. We define $\pi':\tau_{S'}\to\Delta^{\mathcal{I}}$ by $\pi'(y):=\pi(y)$ for all $y\in\tau_S$, $\pi'(y_i):=b_i$ for all $i, 2\leq i\leq m$, and $\pi'(z):=b$. Obviously, $\mathcal{I},\pi'$ satisfy S'.

6. **Number restriction:** Assume that the rule can be applied to $x:(\leq n\ R_1\circ\ldots\circ R_m)\in S$, and let $\mathcal{I}$ together with π be a model of S. On the one hand, since the rule is applicable, x has more than n $R_1\circ\ldots\circ R_m$-successors in S. On the other hand, $\mathcal{I},\pi$ satisfy $x:(\leq m\ R_1\circ\ldots\circ R_m)\in S$, and thus there are two different $R_1\circ\ldots\circ R_m$-successors y_1,y_2 of x in S such that $\pi(y_1)=\pi(y_2)$. Obviously, this implies that $(y_1\neq y_2)\notin S$. Hence $S_{y_1,y_2}=S[y_2/y_1]$ is one of the constraint systems obtained by applying Rule 6 to $x:(\leq n\ R_1\circ\ldots\circ R_m)$. In addition, since $\pi(y_1)=\pi(y_2)$, $\mathcal{I},\pi$ satisfy S_{y_1,y_2}.
Conversely, assume that $S_{y_1,y_2}=S[y_2/y_1]$ is obtained from S by applying Rule 6, and let $\mathcal{I}$ together with the valuation π be a model of S_{y_1,y_2}. If we take a valuation π' that coincides with π on the variables in $\tau_{S_{y_1,y_2}}$ and satisfies $\pi'(y_2)=\pi(y_1)$, then $\mathcal{I},\pi'$ obviously satisfy S. ∎

Proof of Part 2 of Lemma 6.10: Let S be a complete and clash-free constraint system that is obtained by applying the completion rules to $\{x_0:C_0\}$. We define a canonical model $\mathcal{I}$ of S as follows:

$$\begin{array}{rrl} & \Delta^{\mathcal{I}} := \tau_S, & \\ \text{for all } A\in N_C: & x\in A^{\mathcal{I}} & \text{iff } x:A\in S,\\ \text{for all } R\in N_R: & (x,y)\in R^{\mathcal{I}} & \text{iff } xRy\in S.\end{array}$$

In addition, let $\pi:\tau_S\to\Delta^{\mathcal{I}}$ be the identity on τ_S. It remains to show that $\mathcal{I},\pi$ indeed satisfy each constraint in S.

By definition of $\mathcal{I}$, a role constraint of the form xRy is satisfied by $\mathcal{I},\pi$ iff $xRy\in S$. More generally, y is an $R_1\circ\ldots\circ R_m$-successor of x in S iff y is an $R_1\circ\ldots\circ R_m$-successor of x in $\mathcal{I}$. Next, all constraints of the form $x\neq y$ are obviously satisfied since π is the identity.

We show by induction on the structure of the concept C that every concept constraint $x:C\in S$ is satisfied by $\mathcal{I},\pi$. The induction base and the treatment of the propositional constructors and the value restrictions is just as for $\mathcal{ALC}$: $\mathcal{I}$ satisfies all constraints of the form $x : A$ by definition. Since S is clash-free, all constraints of the form $x : \neg A$ are satisfied. By induction, completeness of S implies that S satisfies all constraints of the form $x : C \sqcap D$, $x : C \sqcup D$, $x : \exists R.C$, and $x : \forall R.C$. Again, we go into detail only for constraints involving number restrictions:

- Consider $x : (\geq n\ R_1 \circ \ldots \circ R_m) \in S$. Since S is complete, Rule 5 cannot be applied to $x : (\geq n\ R_1 \circ \ldots \circ R_m)$, and thus x has at least n $R_1 \circ \ldots \circ R_m$-successors in S, which are also $R_1 \circ \ldots \circ R_m$-successors of x in $\mathcal{I}$. This shows that $\mathcal{I}, \pi$ satisfy $x : (\geq n\ R_1 \circ \ldots \circ R_m)$.
- Constraints of the form $x : (\leq n\ R_1 \circ \ldots \circ R_m) \in S$ are satisfied because S is clash-free and complete. In fact, assume that x has more than n $R_1 \circ \ldots \circ R_m$-successors in $\mathcal{I}$. Then x also has more than n $R_1 \circ \ldots \circ R_m$-successors in S. If S contained inequality constraints $y_i \neq y_j$ for all these successors, then we would have a clash. Otherwise, Rule 6 could be applied. ∎

Proof of Part 3 of Lemma 6.10: Assume that S contains a clash. If $\{x : A, x : \neg A\} \subseteq S$, then it is clear that no interpretation can satisfy both constraints. Thus assume that $x : (\leq n\ R_1 \circ \ldots \circ R_m) \in S$ and x has $\ell > n$ $R_1 \circ \ldots \circ R_m$-successors $y_1 \circ \ldots \circ y_\ell$ in S with $(y_i \neq y_j) \in S$ for all $i \neq j$. Obviously, this implies that in any model $\mathcal{I}, \pi$ of S, $\pi(x)$ has $\ell > n$ distinct $R_1 \circ \ldots \circ R_m$-successors $\pi(y_1) \circ \ldots \circ \pi(y_\ell)$ in $\mathcal{I}$, which shows that $\mathcal{I}, \pi$ cannot satisfy $x : (\leq n\ R_1 \circ \ldots \circ R_m)$. ∎

Proof of Part 4 of Lemma 6.10: In the following, we consider only constraint systems S that are obtained by applying the completion rules to $\{x_0 : C_0\}$. For a concept C, we define its and/or-size $|C|_{\sqcap,\sqcup}$ as the number of occurrences of conjunction and disjunction constructors in C. The maximal role depth $\mathsf{depth}(C)$ of C is defined as follows:

$$
\begin{aligned}
&\mathsf{depth}(A) := \mathsf{depth}(\neg A) := 0 \text{ for } A \in N_C,\\
&\mathsf{depth}(C_1 \sqcap C_2) := \mathsf{max}\{\mathsf{depth}(C_1), \mathsf{depth}(C_2)\},\\
&\mathsf{depth}(C_1 \sqcup C_2) := \mathsf{max}\{\mathsf{depth}(C_1), \mathsf{depth}(C_2)\},\\
&\mathsf{depth}(\forall R.C) := \mathsf{depth}(\exists R.C) := 1 + \mathsf{depth}(C),\\
&\mathsf{depth}(\geq n\ R_1 \circ \ldots \circ R_m) := m,\\
&\mathsf{depth}(\leq n\ R_1 \circ \ldots \circ R_m) := m.
\end{aligned}
$$

For the termination proof, the following observations, which are an easy consequence of the definition of the completion rules, are important:

Lemma 6.12 Let S be a constraint system obtained by application of the completion algorithm to $\{x_0 : C_0\}$.

1. Every concept C that may occur in S is a subconcept of C_0.
2. Every variable $x \neq x_0$ that occurs in S is an $R_1 \circ \ldots \circ R_m$-successor of x_0 for some role chain of length $m \geq 1$. In addition, every other role chain that connects x_0 with x has the same length.
3. If x can be reached in S by a role chain of length m from x_0, then for each constraint $x : C$ in S, the maximal role depth of C is bounded by the maximal role depth of C_0 minus m. Consequently, m is bounded by the maximal role depth of C_0.

Proof:

1. is obvious.
2. Suppose the algorithm is started with $\{x_0 : C_0\}$. Then each variable introduced by the algorithm is generated by Rule 4 or 5 and thus connected to its predecessor via some role. The second fact is due to Rule 6, which identifies only those variables that are both $R_1 \circ \ldots \circ R_m$-successors of some y. Lemma 6.12.2 follows then by induction.
3. A constraint $x : C$ can only be present in S if it was added by Rules 1-4. If $x : C$ was added by Rule 1 or 2, then the corresponding superconcept $x : C \sqcap D$ respectively $x : C \sqcup D$ is also present in S, and it is of the same depth. If it was added by Rule 3 or 4, it implies the presence of some $w_i : \exists R.C$ or $w_i : \forall R.C$ in S for some role name R and some R-predecessor w_i of x in S. In both cases, the role depth of the concepts occurring in constraints on w_i are at least 1+ the role depth of C. Hence the maximal role depth of all constraints on all role-predecessors of x is at least 1+ the maximal role depth of all constraints on x. Finally, all variables in τ_S are role successor of C_0, and together with Lemma 6.12.2 this yields Lemma 6.12.3. ∎

Let m_0 be the maximal role depth of C_0. Because of the second fact, every individual x in a constraint system S (reached from $\{x_0 : C_0\}$ by applying completion rules) has a unique role level $\mathsf{level}(x)$, which is its distance from the root node x_0, i.e., the unique length of the role chains that connect x_0 with x. Because of the third fact, the level of each individual is an integer between 0 and m_0.

In the following, we define a mapping κ of constraint systems S to $5(m_0+1)$-tuples of nonnegative integers such that $S \to S'$ implies $\kappa(S) \succ \kappa(S')$, where $\succ$ denotes the lexicographic ordering on $5m_0$-tuples. Since the lexicographic ordering is well-founded, this implies termination of our algorithm. In fact, if the algorithm did not terminate, then there would exist an infinite sequence $S_0 \to S_1 \to \ldots$, and this would yield an infinite descending $\succ$-chain of tuples—in contradiction to the well-foundedness of $\succ$.

Thus, let S be a constraint system that can be reached from $\{x_0 : C_0\}$ by applying completion rules. We define

$$\kappa(S) := (\kappa_0, \kappa_1, \ldots, \kappa_{m_0-1}, \kappa_{m_0}),$$

where $\kappa_\ell := (k_{\ell,1}, k_{\ell,2}, k_{\ell,3}, k_{\ell,4}, k_{\ell,5})$ and the components $k_{\ell,i}$ are obtained as follows:

- $k_{\ell,1}$ is the number of individual variables x in S with $\mathsf{level}(x) = \ell$.
- $k_{\ell,2}$ is the sum of the and/or-sizes $|C|_{\sqcap,\sqcup}$ of all constraints $x : C \in S$ such that $\mathsf{level}(x) = \ell$ and the conjunction or disjunction rule is applicable to $x : C$ in S.
- For a constraint $x : (\geq n\ R_1 \circ \ldots \circ R_m)$, let k be the maximal cardinality of all sets M of $R_1 \circ \ldots \circ R_m$-successors of x for which $y_i \neq y_j \in S$ for all pairs of distinct elements y_i, y_j of M. We associate with $x : (\geq n\ R_1 \circ \ldots \circ R_m)$ the number $r := n - k$, if $n \geq k$, and $r := 0$ otherwise. $k_{\ell,3}$ sums up all the numbers r associated with constraints of the form $x : (\geq n\ R_1 \circ \ldots \circ R_m)$ for variables x with $\mathsf{level}(x) = \ell$.
- $k_{\ell,4}$ is the number of all constraints $x : (\exists R.C) \in S$ such that $\mathsf{level}(x) = \ell$ and the existential restriction rule is applicable to $x : (\exists R.C)$ in S.
- $k_{\ell,5}$ is the number of all pairs of constraints $x : (\forall R.C)$, $xRy \in S$ such that $\mathsf{level}(x) = \ell$ and the value restriction rule is applicable to $x : (\forall R.C)$, xRy in S.

In the following, we show for each of the rules of Figure 6.7 that $S \to S'$ implies $\kappa(S) \succ \kappa(S')$.

1. **Conjunction:** Assume that the rule is applied to the constraint $x : C_1 \sqcap C_2$, and let S' be the system obtained from S by its application. Let $\ell := \mathsf{level}(x)$.
   First, we compare κ_ℓ and κ'_ℓ, the tuples respectively associated with the level ℓ of x in S and S'. Obviously, the *first components* of κ_ℓ and κ'_ℓ agree since the number of individuals and their levels are not changed. The *second component* of κ'_ℓ is *strictly smaller* than the second component of κ_ℓ: $|C_1 \sqcap C_2|_{\sqcap,\sqcup}$ is removed from the sum, and replaced by a number that is not larger than $|C_1|_{\sqcap,\sqcup} + |C_2|_{\sqcap,\sqcup}$ (depending on whether the top constructor of C_1 and C_2 is disjunction or conjunction, or some other constructor). Since tuples are compared with the lexicographic ordering, a decrease in this component makes sure that it is irrelevant what happens in later components.
   For the same reason, we need not consider tuples κ_m for $m > \ell$. Thus, assume that $m < \ell$. In such a tuple, the first three components are not changed by application of the rule, whereas the remaining two components remain unchanged or decrease. Such a decrease can happen if $\mathsf{level}(y) = m$ and S contains constraints yRx, $y :(\forall R.C_i)$ (or $y :(\exists R.C_i)$) for some $i \in \{1, 2\}$.

2. **Disjunction:** This rule can be treated similar to the conjunction rule.

3. **Value restriction:** Assume that the rule is applied to the constraints $x :(\forall R.C), xRy$, and let S' be the system obtained from S by its application. Let $\ell := \mathsf{level}(x)$. Obviously, this implies that $\mathsf{level}(y) = \mathsf{level}(x) + 1 > \ell$.
   On level ℓ, the first three components of κ_ℓ remain unchanged; the fourth remains the same, or decreases (if S contains constraints zSy and $z :(\exists S.C)$ for an individual z with $\mathsf{level}(z) = \ell$); and the fifth decreases by at least one since the constraints $x :(\forall R.C), xRy$ are no longer counted. It may decrease by more than one if S contains constraints zSy and $z :(\forall S.C)$ for an individual z with $\mathsf{level}(z) = \ell$.
   Because of this decrease at level ℓ, the tuples at larger levels (in particular, the one for level $\mathsf{level}(x) + 1$, where there might be an increase), need not be considered.
   The tuples of levels smaller than ℓ are not changed by application of the rule. In particular, the third component of such a tuple does not change since no role constraints or inequality constraints are added or removed.

4. **Existential restriction:** Assume that the rule is applied to the constraint $x :(\exists R.C)$, and let $S' = S \cup \{xRy, y : C\}$ be the system obtained from S by its application. Let $\ell := \mathsf{level}(x)$. Obviously, this implies that

$\mathsf{level}(y) = \mathsf{level}(x) + 1 > \ell$.
The first two components of κ_ℓ obviously remain unchanged. The third component may decrease (if y is the first successor which is introduced for an at-least restriction $(\geq n\ R)$) or it stays the same. Since the fourth component decreases, the possible increase of the fifth component is irrelevant.
For the same reason, the increase of the first component of $\kappa_{\ell+1}$ is irrelevant.
Tuples of a level smaller than ℓ are not increased by the application of this rule. All components of such a tuple remain unchanged, with the possible exception of the third component, which may decrease.

5. **Number restriction:** Assume that the rule is applied to the constraint $x{:}(\geq n\ R_1 \circ \ldots \circ R_m) \in S$, let S' be the system obtained by rule application, and let $\ell = \mathsf{level}(x)$.
   As for Rule 4, the first two components of κ_ℓ remain the same. In addition, there is a decrease in the third component of κ_ℓ, since the new individual z can now be added to the maximal sets of explicitly distinct $R_1 \circ \ldots \circ R_m$-successors of x. Note that these sets were previously smaller than n (because even the set of all $R_1 \circ \ldots \circ R_m$-successors of x was smaller than n).
   For this reason, the possible increase in the fifth component of κ_ℓ and in the first components of tuples of levels larger than ℓ are irrelevant. Tuples of a level smaller than ℓ are either unchanged by application of the rule, or their third component decreases.

6. **Number restriction:** Assume that the rule is applied to the constraint $x{:}(\leq n\ R_1 \circ \ldots \circ R_m) \in S$, let $S' = S_{y_1,y_2}$ be the system obtained by rule application, and let $\ell = \mathsf{level}(x)$.
   On level $\ell + m$, the first component of the tuple $\kappa_{\ell+m}$ decreases. Thus, possible increases in the other components of this tuple are irrelevant.
   Tuples associated with smaller levels remain unchanged or decrease. In fact, since y_1 in S' has all its old constraints and the constraints of y_2 in S, some value restrictions or existential restrictions for individuals of the level immediately above level $\ell + m$ may become satisfied (in the sense that the corresponding rule no longer applies). Since no constraints are removed, previously satisfied value restrictions or existential restrictions remain satisfied. The third component of tuples of smaller level cannot increase since the individuals y_1, y_2 that have been identified were not related by inequality constraints. ∎

For languages where number restrictions may contain—in addition to composition—union or intersection of roles, an important property used in the above termination proof is no longer given: It is not possible to associate each individual generated by a tableau-based algorithm with a unique role level, which is its distance to the "root" individual x_0 (i.e., the instance x_0 of C_0 generated by the tableau algorithm). Indeed, in the concept

$$C_0 := (\exists R.\exists R.A) \sqcap (\leq 1\ R \sqcup R \circ R),$$

the number restriction enforces that an R-successor of an instance of C_0 is also an $R \circ R$-successor of this instance. For this reason, an R-successor of the root individual must be both on level 1 and on level 2, and thus the relatively simple termination argument that was used above is not available for these larger languages. However, as we shall show in the next section, this termination argument can still be used if union and intersection are restricted to role chains of the same length. Without this restriction, satisfiability may become undecidable: in Section 6.1.1, we have shown that satisfiability is in fact undecidable for $\mathcal{ALCN}(\circ, \sqcap)$. For $\mathcal{ALCN}(\circ, \sqcup)$, decidability of satisfiability is still an open problem.

An extension of the decidability result

The algorithm given in Section 6.1.2 can be extended such that it can also treat union and intersection of role chains that have the same length. The proof of soundness, completeness and termination of this extended algorithm is very similar to the one for the basic algorithm, and will thus only be sketched.

In the remainder of this section, a *complex role* is

- a role chain $\mathbf{R} = R_1 \circ \ldots \circ R_n$, or
- the intersection $\mathbf{R} = R_1 \circ \ldots \circ R_n \sqcap S_1 \circ \ldots \circ S_n$ of two role chains of the same length, or
- the union $\mathbf{R} = R_1 \circ \ldots \circ R_n \sqcup S_1 \circ \ldots \circ S_n$ of two role chains of the same length.

The satisfiability algorithm is extended by adding two new rules 5a and 5b to handle number restrictions $(\geq n\ \mathbf{R})$ for complex roles with union or intersection, and by substituting rule 6 by a new rule 6′ that is able to handle the new types of complex roles. To formulate the new rules, we must extend the notion of a role successor in a constraint system appropriately. Building up on the notion of a role successor for a role chain (see Definition 4.11), we define:

5a. Number restriction: If $x:(\geq n\ R_1\circ\ldots\circ_m \sqcup S_1\circ\ldots\circ S_m) \in S$ and x has less than n $(R_1\circ\ldots\circ R_m \sqcup S_1\circ\ldots\circ S_m)$-successors in S, then
$S \to S_1 = S \cup \{xR_1y_2, y_mR_mz\} \cup \{y_iR_iy_{i+1} \mid 2 \leq i \leq m-1\} \cup$
$\{z \neq w \mid w \text{ is an } (R_1\circ\ldots\circ R_m \sqcup S_1\circ\ldots\circ S_m)\text{-successor of } x \text{ in } S\}$
$S \to S_2 = S \cup \{xS_1y_2, y_mS_mz\} \cup \{y_iS_iy_{i+1} \mid 2 \leq i \leq m-1\} \cup$
$\{z \neq w \mid w \text{ is an } (R_1\circ\ldots\circ R_m \sqcup S_1\circ\ldots\circ S_m)\text{-successor of } x \text{ in } S\}$
where z, y_i are new variables in $\tau \setminus \tau_S$.

5b. Number restriction: If $x:(\geq n\ R_1\circ\ldots\circ R_m \sqcap S_1\circ\ldots\circ S_m) \in S$ and x has less than n $(R_1\circ\ldots\circ R_m \sqcap S_1\circ\ldots\circ S_m)$-successors in S, then
$S \to\ S \cup \{xR_1y_2, xS_1y_2', y_mR_mz, y_m'S_mz\} \cup$
$\{y_iR_iy_{i+1}, y_i'S_iy_{i+1}' \mid 2 \leq i \leq m-1\} \cup$
$\{z \neq w \mid w \text{ is an } (R_1\circ\ldots\circ R_m \sqcap S_1\circ\ldots\circ S_m)\text{-successor of } x \text{ in } S\}$
where z, y_i', y_i are new variables in $\tau \setminus \tau_S$.

6'. Number restriction: If $x:(\leq n\ \mathbf{R}) \in S$ for some complex role $\mathbf{R}$, x has more than n $\mathbf{R}$-successors in S, and there are $\mathbf{R}$-successors y_1, y_2 of x in S with $(y_1 \neq y_2) \notin S$, then
$S \to\ S_{y_1,y_2} = S[y_2/y_1]$
for all pairs y_1, y_2 of $\mathbf{R}$-successors of x with $(y_1 \neq y_2) \notin S$.

Figure 6.8: The additional completion rules.

- y is an $(R_1\circ\ldots\circ R_n \sqcup S_1\circ\ldots\circ S_n)$-successor of x in S iff y is an $R_1\circ\ldots\circ R_n$-successor or an $S_1\circ\ldots\circ S_n$-successor of x in S, and
- y is an $(R_1\circ\ldots\circ R_n \sqcap S_1\circ\ldots\circ S_n)$-successor of x in S iff y is an $R_1\circ\ldots\circ R_n$-successor and an $S_1\circ\ldots\circ S_n$-successor of x in S.

Obviously, this definition is such that role successors in S are also role successors in every model of S: if $\mathcal{I}, \pi$ satisfy S, and y is an $\mathbf{R}$-successor of x in S for a complex role $\mathbf{R}$, then $\pi(y)$ is an $\mathbf{R}$-successor of $\pi(x)$ in $\mathcal{I}$.

The new rules are described in Figure 6.8. The rules $5a$, $5b$ are added to the completion rules, whereas rule $6'$ substitutes rule 6 in Figure 6.7. To show that the new algorithm obtained this way decides satisfiability of concepts for the extended language, we must proof that all four parts of Lemma 6.10 still hold. The observations presented in Lemma 6.12 still hold, only in the part 2 of Lemma 6.12 one has to substitute "due to Rule 6 which identifies only those variables that are both $R_1 \circ \ldots \circ R_m$-successors of some y" by "due to Rule 6′ which identifies only those variables that are both successors of some y with respect to role chains having the same length".

1. *Local correctness* of the rules $5a, 5b$ and $6'$ can be shown as in the proof of Part 1 of Lemma 6.10 above.

2. The *canonical model* induced by a complete and clash-free constraint system is defined as in the proof of Part 2 of Lemma 6.10. The proof that this canonical model really satisfies the constraint system is also similar to the one given there. Note that the notion of an $\mathbf{R}$-successor of a complex role $\mathbf{R}$ in a constraint system was defined such that it coincides with the notion of an $\mathbf{R}$-successor in the canonical model $\mathcal{I}$ induced by the constraint system.

3. The proof that a constraint system containing a clash is unsatisfiable is the same as the one given above. Note that this depends on the fact that role successors in a constraint system are also role successors in every model of the constraint system.

4. The proof of *termination* is also very similar to the one given above. The definition of the depth of a concept is extended in the obvious way to concepts with number restrictions on complex roles:

$$\begin{aligned}
\mathsf{depth}(\geq n\ R_1 \circ \ldots \circ R_m \sqcap S_1 \circ \ldots \circ S_m) &:= m,\\
\mathsf{depth}(\geq n\ R_1 \circ \ldots \circ R_m \sqcup S_1 \circ \ldots \circ S_m) &:= m,\\
\mathsf{depth}(\leq n\ R_1 \circ \ldots \circ R_m \sqcap S_1 \circ \ldots \circ S_m) &:= m,\\
\mathsf{depth}(\leq n\ R_1 \circ \ldots \circ R_m \sqcup S_1 \circ \ldots \circ S_m) &:= m.
\end{aligned}$$

   Because the role chains in complex roles are of the same length, it is easy to see that Lemma 6.12 still holds. Thus, we can define the same measure $\kappa(S)$ as above for all constraint systems obtained by applying the extended completion rules to $\{x_0 : C_0\}$. It is easy to see that the proof that $S \to S'$ implies $\kappa(S) \succ \kappa(S')$ can be extended to the new rules. It should be noted that the proof given above is already formulated in a more general way than it were necessary for $\mathcal{ALCN}(\circ)$. In fact, we have only used the fact that all role chains connecting two individuals have the same length (which is still satisfied for the extended language), and the fact that these role chains also have the same name was not mentioned (which is only satisfied for $\mathcal{ALCN}(\circ)$).

The following theorem is an immediate consequence of these observations:

Theorem 6.13 Subsumption and satisfiability is decidable for the language that extends $\mathcal{ALCN}(\circ)$ by number restrictions on union and intersection of role chains of the same length.

Given this decidability result, a natural question arising here is whether consistency of $\mathcal{ALCN}(\circ)$-ABoxes is also decidable. In Section 4.3, we already mentioned that a tableau-based algorithm deciding satisfiability of concepts is likely to be extendible such that it decides ABox consistency. Unfortunately, this is not true for $\mathcal{ALCN}(\circ)$-ABoxes. In [Molitor1997], the failure of several attempts to design a decision procedure for the consistency of $\mathcal{ALCN}(\circ)$-ABoxes is described. All these attempts led either to non-termination or incompleteness. A decidability result could only be obtained for ABoxes of a severely restricted form. Roughly spoken, consistency of $\mathcal{ALCN}(\circ)$-ABoxes can only be decided for ABoxes which do not contain two individuals which are connected via two roles chains of different lengths.

6.2 Symbolic Number Restrictions

As in the last section, we first give the undecidability result: This yields a better insight into the expressive power of $\mathcal{ALCN}^S$, and it motivates the investigation of the restricted language $\mathcal{ALUEN}^S$.

6.2.1 An undecidability result

As in Section 6.1.1, undecidability of satisfiability is shown by a reduction of the domino problem to concept satisfiability. For $\mathcal{ALCN}^S$, however, the proof is easier if we take another variant of the domino problem: Instead of asking for a compatible tiling of the first quadrant of the plane, we ask now for a compatible tiling of the "second eighth"

$$(\mathbb{N} \times \mathbb{N})_{\leq} := \{(a, b) \mid a, b \in \mathbb{N} \text{ and } a \leq b\}$$

of the plane. As a consequence, the compatibility condition of a tiling is less strict than the one for a tiling of the first quadrant of the plane. More precisely, a tiling $t : (\mathbb{N} \times \mathbb{N})_{\leq} \times (\mathbb{N} \times \mathbb{N})_{\leq} \to D$ is now called compatible iff for all $m, n \in \mathbb{N}$:

$$m < n \Rightarrow (t(m, n), t(m + 1, n)) \in H \text{ and } (t(m, n), t(m, n + 1)) \in V.$$

As such a tiling yields compatible tilings of arbitrarily large finite rectangles, it also yields a compatible tiling of the plane [Knuth1968].

In contrast to the reduction given in Section 6.1.1, in this reduction, the individuals representing points in the grid are not related to each other by roles—there is nothing comparable to the "horizontal" and "vertical" roles X and Y. Intuitively, the new reduction works as follows: First, we define an $\mathcal{ALCN}^S$-concept C_{N} such that, for each model of C_{N} with $x \in C_{\mathrm{N}}^{\mathcal{I}}$, there is a natural relationship between tuples $(a, b) \in (\mathbb{N} \times \mathbb{N})_{\leq}$ and S-successors $y_{a,b}$ of x. The point (a, b) is represented by an S-successor of x having a L-successors and b R-successors. Second, for a given tiling system $\mathcal{D}$, we construct a concept $C_{\mathcal{D}}$ that

1. is subsumed by C_{N},
2. ensures that every $y_{a,b}$ is associated to exactly one domino type, and
3. ensures that the horizontal and vertical matching conditions are satisfied.

The formal definition of C_{N} is given in the upper part of Figure 6.9. Let $\mathcal{I}$ be a model of C_{N} with $x \in C_{\mathrm{N}}^{\mathcal{I}}$. Now, C_1 expresses that for every nonnegative integer a, x has an S-successor having exactly a L-successors. The precondition of C_2 makes sure that a is smaller than b, and thus the whole implication says that for each pair a, b of nonnegative integers, if $a \leq b$ then x has an S-successor having exactly a L-successors and b R-successors (there can be more than one such S-successor). Finally, C_3 says that whenever an S-successor of x has a L-successors and b R-successors, we have $a \leq b$. Thus, there is an obvious correspondence between S-successors of x and points in the second eighth of the plane: every S-successor corresponds to a point in $(\mathbb{N} \times \mathbb{N})_{\leq}$ and vice versa. To be more precise, we will formally prove the following observations concerning C_{N}. For a role name R and some $x \in \Delta^{\mathcal{I}}$, ${x_R}^{\mathcal{I}}$ denotes the number of role fillers of x with respect to R in $\mathcal{I}$, i.e.

$$
{x_R}^{\mathcal{I}} := \#\{y \in \Delta^{\mathcal{I}} \mid (x, y) \in R^{\mathcal{I}}\}.
$$

Lemma 6.14 1. C_{N} is satisfiable.

2. Let $\mathcal{I}$ be a model of C_{N} with $x \in C_{\mathrm{N}}^{\mathcal{I}}$ and let

$$
Y_x = \{y \in \Delta^{\mathcal{I}} \mid (x, y) \in S^{\mathcal{I}}\}.
$$

(i) For each $(a, b) \in (\mathbb{N} \times \mathbb{N})_{\leq}$ there exists $y_{a,b} \in Y_x$ with ${(y_{a,b})_L}^{\mathcal{I}} = a$ and ${(y_{a,b})_R}^{\mathcal{I}} = b$.

$C_{\mathrm{N}} := (\uparrow\alpha.\uparrow\beta.(C_1 \sqcap C_2 \sqcap C_3))$ where

$$C_1 := (\exists S.(= \alpha\ L))$$
$$C_2 := ((\exists S.(= \alpha\ L) \sqcap (\leq \beta\ L)) \Rightarrow (\exists S.(= \alpha\ L) \sqcap (= \beta\ R)))$$
$$C_3 := (\forall S.((= \alpha\ L) \sqcap (= \beta\ R)) \Rightarrow (\leq \beta\ L))$$

Given a tiling system $\mathcal{D} = (\{D_1, \ldots, D_m\}, H, V)$ and the subconcepts C_1, C_2, C_3 of C_{N} as defined above, let

$$\begin{array}{lll}
C_{\mathcal{D}} := C_{\mathrm{N}} \sqcap (\forall S.(\bigsqcup_{1\leq i\leq m}(D_i \sqcap (\bigsqcap_{1\leq j\leq m, i\neq j} \neg D_j)))) \sqcap & & \\
\quad (\uparrow\alpha.\uparrow\beta. & & \\
\quad \bigsqcap_{1\leq i\leq m}(\exists S.((= \alpha\ L) \sqcap (= \beta\ R) \sqcap D_i)) \Rightarrow & & \\
\quad (1) & ((\forall S.((\neq \alpha\ L) \sqcup (\neq \beta\ R) \sqcup D_i)) \sqcap & \\
\quad (2) & (\uparrow\gamma.({<}(\alpha,\beta) \sqcap {=}(\alpha+1,\gamma)) \Rightarrow & \\
 & \quad (\forall S.(((= \gamma\ L) \sqcap (= \beta\ R)) \Rightarrow \bigsqcup_{(D_i,D_j)\in H} D_j))) \sqcap & \\
\quad (3) & (\uparrow\gamma.({=}(\beta+1,\gamma) \Rightarrow & \\
 & \quad (\forall S.(((= \alpha\ L) \sqcap (= \gamma\ R)) \Rightarrow \bigsqcup_{(D_i,D_j)\in V} D_j)))))) &
\end{array}$$

where the following abbreviations are used

$$\begin{aligned}
{<}(\alpha,\beta) &:= (\exists S.((= \alpha\ L) \sqcap (= \beta\ R) \sqcap \neg(= \beta\ L))),\\
{=}(\alpha+1,\beta) &:= {<}(\alpha,\beta) \sqcap (\forall S.((\leq \alpha\ L) \sqcup (\geq \beta\ L))).
\end{aligned}$$

Figure 6.9: Reduction concepts used in the proof of Theorem 6.16.

(ii) If $y \in Y_x$ and $y_L{}^{\mathcal{I}} = a$ and $y_R{}^{\mathcal{I}} = b$, then $(a, b) \in (\mathbb{N} \times \mathbb{N})_{\leq}$.

3. If $x \in C_{\mathrm{N}}{}^{\mathcal{I}}$, then there is an injective mapping $\phi : (\mathbb{N} \times \mathbb{N})_{\leq} \to Y_x$ from the second eighth of the plane to the set of S-successors of x.

Proof: 1. Define $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ and x as follows:

$$\begin{aligned}
\Delta^{\mathcal{I}} &= \{x\} \uplus \{y_{a,b} \mid (a,b) \in (\mathbb{N} \times \mathbb{N})_{\leq}\} \uplus \{l_a, r_b \mid a, b \in \mathbb{N}\},\\
S^{\mathcal{I}} &= \{(x, y_{a,b}) \mid (a,b) \in (\mathbb{N} \times \mathbb{N})_{\leq}\},\\
L^{\mathcal{I}} &= \{(y_{a,b}, l_{a'}) \mid (a,b) \in (\mathbb{N} \times \mathbb{N})_{\leq} \text{ and } a' < a\},\\
R^{\mathcal{I}} &= \{(y_{a,b}, r_{b'}) \mid (a,b) \in (\mathbb{N} \times \mathbb{N})_{\leq} \text{ and } b' < b\}.
\end{aligned}$$

$\mathcal{I}$ is an $\mathcal{ALCN}^S$-interpretation and $L^{\mathcal{I}}, R^{\mathcal{I}}$ are defined such that, for all $(a, b) \in (\mathbb{N} \times \mathbb{N})_{\leq}$, we have $(y_{a,b})_L{}^{\mathcal{I}} = a$ and $(y_{a,b})_R{}^{\mathcal{I}} = b$. We show that $x \in C_{\mathrm{N}}{}^{\mathcal{I}}$:

$x \in C_{\mathbb{N}}{}^{\mathcal{I}}$ iff for all $a, b \in \mathbb{N}$: $x \in (C_1[\alpha/a][\beta/b])^{\mathcal{I}}$, $x \in (C_2[\alpha/a][\beta/b])^{\mathcal{I}}$ and $x \in (C_3[\alpha/a][\beta/b])^{\mathcal{I}}$. If $a, b \in \mathbb{N}$, then

- $x \in (C_1[\alpha/a][\beta/b])^{\mathcal{I}}$ since $(x, y_{a,b'}) \in S^{\mathcal{I}}$ for some (even all) b' with $b' \geq a$.
- $x \in (C_2[\alpha/a][\beta/b])^{\mathcal{I}}$: If $x \in (\exists S.(= a\ L) \sqcap (\leq b\ L))^{\mathcal{I}}$, then $a \leq b$ and $(x, y_{a,b}) \in S^{\mathcal{I}}$, which implies $x \in (\exists S.(= a\ L) \sqcap (= b\ R))^{\mathcal{I}}$.
- $x \in (C_3[\alpha/a][\beta/b])^{\mathcal{I}}$: Let $(x, y) \in S^{\mathcal{I}}$. If $y \in ((= a\ L) \sqcap (= b\ R))^{\mathcal{I}}$, then $y = y_{a,b}$ with $a \leq b$, which implies $y \in (\leq b\ L)^{\mathcal{I}}$.

2. (i) The subconcept C_1 makes sure that for each $a \in \mathbb{N}$, there exists some $y \in Y_x$ with $y_L{}^{\mathcal{I}} = a$. C_2 ensures for all $a, b \in \mathbb{N}$ that, if $y \in Y_x$ with $y_L{}^{\mathcal{I}} = a$ and $y_L{}^{\mathcal{I}} \leq b$ (which holds for all b with $a \leq b$), then there is some $y' \in Y_x$ with $y_L'{}^{\mathcal{I}} = a$ and $y_R'{}^{\mathcal{I}} = b$.

2. (ii) The third subconcept C_3 enforces for all $y \in Y_x$ that $y_L{}^{\mathcal{I}} = a$ and $y_R{}^{\mathcal{I}} = b$ implies $y_L{}^{\mathcal{I}} \leq b$, which yields $a \leq b$.

3. For $a, b \in (\mathbb{N} \times \mathbb{N})_{\leq}$ there can be more than one $y \in Y_x$ with $y_L{}^{\mathcal{I}} = a$ and $y_R{}^{\mathcal{I}} = b$. Hence, in order to define an injective mapping, we assume that Y_x is linearly ordered by some ordering $\prec$. Then ϕ is defined as follows:

$$\begin{array}{rlll} \phi : & (\mathbb{N} \times \mathbb{N})_{\leq} & \rightarrow & Y_x \\ & (a, b) & \mapsto & \mathsf{min}_{\prec}\{y \in Y_x \mid y_L{}^{\mathcal{I}} = a \text{ and } y_R{}^{\mathcal{I}} = b\} \end{array}$$

where, for $M \subseteq Y_x$, $\mathsf{min}_{\prec} M$ denotes the minimum of M with respect to $\prec$. ϕ is injective by definition, and it is a total mapping as a direct consequence of 2. ∎

The definition of the concept $C_{\mathcal{D}}$ associated with a tiling system $\mathcal{D}$ is also given in Figure 6.9. In the definition, two abbreviations $<$ and $=$ are used. In the context of the concept $C_{\mathbb{N}}$, these abbreviations really express the relation $<$ and the successor relation on natural numbers: For $x \in C_{\mathbb{N}}{}^{\mathcal{I}}$ we have

$$\text{“}x \in ({<}(\alpha, \beta)[\alpha/a][\beta/b])^{\mathcal{I}} \text{ iff } a < b\text{”}$$

as an immediate consequence of Lemma 6.14.2. Furthermore, we have

$$\text{“}x \in ({=}(\alpha + 1, \beta)[\alpha/a][\beta/b])^{\mathcal{I}} \text{ iff } a + 1 = b\text{”}$$

because x has some S-successor having a L-successors for each $a \in \mathbb{N}$. The definitions of the relation $<$ and the successor relation on natural numbers are so easy because we defined $C_{\mathbb{N}}$ in such a way that Y_x corresponds to $(\mathbb{N} \times \mathbb{N})_{\leq}$ in the way mentioned in Lemma 6.14, namely because each S-successor y of x has at most $y_R{}^{\mathcal{I}}$ L-successors. Replacing $(\mathbb{N} \times \mathbb{N})_{\leq}$ by $\mathbb{N} \times \mathbb{N}$ (thus using the first quadrant of the plane instead of the second octant) would have made the definition of the successor relation on natural numbers far more complicated.

The first line in the definition of $C_{\mathcal{D}}$ makes sure that $C_{\mathbb{N}}$ subsumes $C_{\mathcal{D}}$, and that every S-successor of an instance x of $C_{\mathcal{D}}$ has exactly one domino type. In the remainder of the definition, we consider an S-successor $y_{a,b}$ of x with domino type D_i that has a L- and b R-successors. Now, the subconcept (1) of $C_{\mathcal{D}}$ ensures that every S-successor with the same number of L- and R-successors as $y_{a,b}$ has the same domino type D_i. (2) takes care of the horizontal matching condition. The conjunct $<(\alpha, \beta)$ is found in (2) because the horizontal matching condition asks only for matching right neighbours of points (a, b) with $a < b$, whereas the domino type associated to a point (a, a) does not impose any constraints on the domino type associated to $(a + 1, a)$. Finally, (3) takes care of the vertical matching condition.

Lemma 6.15 $C_{\mathcal{D}}$ is satisfiable iff $\mathcal{D}$ allows for a compatible tiling.

Proof: From the definition of $C_{\mathcal{D}}$ it follows immediately that $C_{\mathcal{D}}$ is subsumed by $C_{\mathbb{N}}$.

"⇒" Given a model $\mathcal{I}$ of $C_{\mathcal{D}}$ with $x \in C_{\mathcal{D}}{}^{\mathcal{I}}$, we define a mapping $t : (\mathbb{N} \times \mathbb{N})_{\leq} \to D$ as follows:

$$t(a, b) = D_i \text{ iff } x \in (\exists S.((= a\ L) \sqcap (= b\ R) \sqcap D_i))^{\mathcal{I}}.$$

First, we show that t is well-defined: Let $a, b \in \mathbb{N}$. Since

$$x \in (\forall S.(\bigsqcup_{1 \leq i \leq m} (D_i \sqcap (\bigsqcap_{1 \leq j \leq m, i \neq j} \neg D_j))))^{\mathcal{I}},$$

each S-successor of x is an instance of exactly one $D_i \in D$. Furthermore, for each $(a, b) \in (\mathbb{N} \times \mathbb{N})_{\leq}$ and for each $D_i \in D$

$$x \in ((\exists S.((= a\ L) \sqcap (= b\ R) \sqcap D_i)) \Rightarrow (\forall S.((\neq a\ L) \sqcup (\neq b\ R) \sqcup D_i)))^{\mathcal{I}},$$

hence all S-successors of x having the same number of L-successors and the same number of R-successors are instances of the same $D_i \in D$. Finally, as a consequence of Lemma 6.14, x has an S-successor y with

${y_L}^{\mathcal{I}} = a$ and ${y_R}^{\mathcal{I}} = b$ for each $(a,b) \in (\mathbb{N} \times \mathbb{N})_{\leq}$. Thus t is well-defined, and it remains to be shown that t is indeed compatible:

Let $a, b \in \mathbb{N}$ with $a < b$, let y be an S-successor of x with ${y_L}^{\mathcal{I}} = a$ and ${y_R}^{\mathcal{I}} = b$, and let $y \in D_i$ (hence we have $t(a,b) = D_i$). From $x \in {C_{\mathcal{D}}}^{\mathcal{I}}$ it follows that

$$x \in (\uparrow\gamma.((<(a,b) \sqcap =(a+1,\gamma)) \Rightarrow (\exists S.((=\gamma\ L) \sqcap (=b\ R) \sqcap D_j))))^{\mathcal{I}}$$

for some D_j with $(D_i, D_j) \in H$. Hence $x \in (\exists S.((=a+1\ L) \sqcap (= b\ R) \sqcap D_j))^{\mathcal{I}}$, which implies that $t(a+1,b) = D_j$ with $(D_i, D_j) \in H$.

The same arguments apply to the vertical matching condition, which shows that t is indeed a compatible tiling.

"$\Leftarrow$" Given a compatible tiling t, a model $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ of $C_{\mathcal{D}}$ can be defined like the one for C_{N} in the proof of Lemma 6.14, i.e.,

$$\begin{array}{lcl}
\Delta^{\mathcal{I}} & := & \{x\} \uplus \{y_{a,b} \mid a,b \in \mathbb{N} \text{ and } a \leq b\} \uplus \{l_a, r_b \mid a, b \in \mathbb{N}\}, \\
S^{\mathcal{I}} & := & \{(x, y_{a,b}) \mid a,b \in \mathbb{N} \text{ and } a \leq b\}, \\
L^{\mathcal{I}} & := & \{(y_{a,b}, l_{a'}) \mid a, a', b \in \mathbb{N} \text{ and } a' < a \leq b\}, \\
R^{\mathcal{I}} & := & \{(y_{a,b}, r_{b'}) \mid a, b, b' \in \mathbb{N} \text{ and } a \leq b \text{ and } b' < b\}.
\end{array}$$

In addition,

$$D_i^{\mathcal{I}} \quad := \quad \{y_{a,b} \mid t(a,b) = D_i\}.$$

Each S-successor of x is instance of exactly one $D_i \in D$, hence

$$x \in (\forall S.(\bigsqcup_{1 \leq i \leq m} (D_i \sqcap (\bigsqcap_{1 \leq j \leq m_{\bar{i}} \neq j} \neg D_j))))^{\mathcal{I}}.$$

The proof of Lemma 6.14 shows that $x \in C_{\mathrm{N}}^{\mathcal{I}}$. Now let $a, b, g \in \mathbb{N}$. Then $x \in ((\exists S.((=\alpha\ L) \sqcap (=\beta\ R) \sqcap D_i))[\alpha/a][\beta/b])^{\mathcal{I}}$ iff $a \leq b$ and $t(a,b) = D_i$. If $x \in ((\exists S.((=\alpha\ L) \sqcap (=\beta\ R) \sqcap D_i))[\alpha/a][\beta/b])^{\mathcal{I}}$, then

- $x \in (\forall S.((\neq \alpha\ L) \sqcup (\neq \beta\ R) \sqcup D_i)[\alpha/a][\beta/b])^{\mathcal{I}}$ since x has only a single S-successor $y_{a,b} \in \Delta^{\mathcal{I}}$ having a L-successors and b R-successors, and according to the assumption $y_{a,b} \in D_i$.
- if $x \in ((<(\alpha,\beta) \sqcap =(\alpha+1,\gamma))[\alpha/a][\beta/b][\gamma/g])^{\mathcal{I}}$, then $a < b$ and $a+1 = g$, and the definition of $\mathcal{I}$ implies that $x \in (\exists S.((=\gamma\ L) \sqcap (=\beta\ R) \sqcap D_j)[\beta/b][\gamma/g])^{\mathcal{I}}$ for some D_j with $(D_i, D_j) \in H$. Hence

$$\begin{array}{l}
x \in \\
((<(\alpha,\beta) \sqcap =(\alpha+1,\gamma)) \Rightarrow (\exists S.(\,(=\gamma\ L) \sqcap (=\beta\ R) \sqcap \\
\qquad\qquad\qquad\qquad \sqcup_{j \in H(D_i)} D_j))[\alpha/a][\beta/b][\gamma/g])^{\mathcal{I}}.
\end{array}$$

- if $x \in (=(\beta+1,\gamma))[\beta/b][\gamma/g])^{\mathcal{I}}$, then $b+1=g$, and the definition of $\mathcal{I}$ implies that $x \in (\exists S.((= \alpha\ L) \sqcap (= \gamma\ R) \sqcap D_j)[\alpha/a][\gamma/g])^{\mathcal{I}}$ for some D_j with $(D_i, D_j) \in V$. Hence

$$x \in ((=(\beta+1,\gamma)) \Rightarrow (\exists S.(\ (= \alpha\ L) \sqcap (= \gamma\ R) \sqcap \sqcup_{j \in V(D_i)} D_j))[\alpha/a][\beta/b][\gamma/g])^{\mathcal{I}}.$$

Summing up, we have $x \in C_{\mathcal{D}}$. ∎

Thus, undecidability of the domino problem yields undecidability of the satisfiability problem for $\mathcal{ALCN}^S$-concepts. Since C is unsatisfiable iff $C \sqsubseteq (A \sqcap \neg A)$, this implies undecidability of subsumption.

Theorem 6.16 Satisfiability and subsumption of $\mathcal{ALCN}^S$-concepts are undecidable.

6.2.2 A decidability result

In this section, it will be shown that satisfiability of $\mathcal{ALUEN}^S$-concepts is decidable. In order to simplify our investigation of the satisfiability problem for $\mathcal{ALUEN}^S$-concepts, we will restrict our attention to concepts in *variable normal form* (VNF), this is to concepts where each variable is bound at most once by $\downarrow$. Obviously, each $\mathcal{ALUEN}^S$-concept can be transformed to an equivalent concept in VNF by renaming of variables which are bound more than once.

Decidability of satisfiability of $\mathcal{ALUEN}^S$-concepts will be shown by presenting a tableau-based algorithm and showing that for each $\mathcal{ALUEN}^S$-concept C, this algorithm is sound, complete, and terminating. Similar to the algorithm presented in Section 6.1.2, the algorithm works on constraints, but for $\mathcal{ALUEN}^S$-concepts, we need additional variables α_x: Suppose we have the constraint $y:(\forall R.(\downarrow\alpha.C))$. Then, for each R-successor x of y, we need a variable α_x that stand for α "in the context of x".

Definition 6.17 We assume that we have a countably infinite set $\tau = \{x, y, z, \dots\}$ of individual variables, and for each pair $(\alpha, x) \in N_V \times \tau$ a new numerical variable α_x which may occur free in concepts. Constraints, constraint systems and clashes are defined as for $\mathcal{ALC}$ in Definition 4.6.

An interpretation $\mathcal{I}$ is a model of a constraint system S iff there is a mapping $\pi : \tau \rightarrow \Delta^{\mathcal{I}}$ and a mapping $\nu : N_V \times \tau \rightarrow \mathbb{N}$ such that $\mathcal{I}, \pi, \nu$ satisfy each constraint in S, i.e., we have

$(\pi(x), \pi(y)) \in R^{\mathcal{I}}$ for all $xRy \in S$,

$\pi(x) \in \nu(D)^{\mathcal{I}}$ for all $x : D \in S$,

where $\nu(D)$ is obtained from D by replacing each variable α_y by its ν-image $\nu(\alpha, y)$.

In the sequel, rel denotes any relation in $\{\leq, \geq\}$. A constraint system S is said to be *numerically consistent* iff the conjunction of all numerical constraints in S, i.e.,

$$S_{\mathsf{num}} := \bigwedge_{\substack{x : (\mathsf{rel}\, n\ R) \in S \\ x \in \tau, R \in N_R, n \in \mathbb{N}}} (x_R \,\mathsf{rel}\, n) \wedge \bigwedge_{\substack{x : (\mathsf{rel}\, \alpha_y\ R) \in S \\ x, y \in \tau, R \in N_R, \alpha \in N_V}} (x_R \,\mathsf{rel}\, \alpha_y),$$

is satisfiable in $(\mathbb{N}, <)$, where x_R, α_y are interpreted as variables for nonnegative integers.

A constraint system S is called *complete* iff none of the completion rules of Figure 6.10 can be applied to S.

The algorithm works exactly like the tableau-based algorithm presented for $\mathcal{ALC}$ in Section 4.4 with the only differences that

- it uses the completion rules given in Figure 6.10, and
- it answers "C_0 is satisfiable" only if it has generated a complete, clash-free, *and* numerically consistent constraint system. The $\mathcal{ALC}$-algorithm asked only for a complete and clash-free constraint system.

Before showing that this completion algorithm yields a decision procedure for satisfiability of $\mathcal{ALUEN}^S$-concepts, let us make some comments on the rules. First, note that each of the completion rules adds constraints when applied to a constraint system, none of the rules removes constraints, and individual variables $x \in \tau$ are never identified nor substituted. With respect to this last property, our algorithm differs from the tableau-based algorithms for $\mathcal{ALCN}$ described in [Donini *et al.*1991a] and for $\mathcal{ALCN}(\circ)$ presented in the previous section. Unlike our Rule 4, these algorithms introduce for each constraint of the form $x : \exists R.C$ a new R-successor of x. If x also has a constraint of the form $x : (\leq n\ R)$, and more than n R-successors have been introduced, then some of these individuals are identified. Our Rule 4 avoids identification by "guessing" the number of allowed R-successors of x before introducing these successors. In fact, since we have numerical variables beside explicit numbers, and since restrictions on numerical variables α_y in constraints $x:(\leq \alpha_y\ R)$ can derive from different parts of the constraint system, a similar identification on demand would either lead to non-termination or incorrectness. The second new feature is Rule 3. Given a constraint $x:(\downarrow\alpha.D)$, we substitute a new numerical variable α_x for α to make sure that the semantics of the existential quantifier $\downarrow\alpha$ is obeyed, i.e., that the valuation for α depends on x. If we would just use α, the difference between $\downarrow\alpha.\forall R.D$ and $\forall R.(\downarrow\alpha.D)$ would not be captured.

Decidability of satisfiability of $\mathcal{ALUEN}^S$-concepts is an easy consequence of the following lemma.

Lemma 6.18 Let C_0 be an $\mathcal{ALUEN}^S$-concept in VNF, and let S be a constraint system obtained by applying the completion rules to $\{x_0 : C_0\}$. Then

1. The completion algorithm terminates when applied to $\{x_0 : C_0\}$.
2. For each completion rule $\mathcal{R}$ that can be applied to S, and for each interpretation $\mathcal{I}$, (i) and (ii) are equivalent.
   (i) $\mathcal{I}$ is a model of S.
   (ii) $\mathcal{I}$ is a model of one of the systems S_i obtained by applying $\mathcal{R}$.
3. If S is a clash-free, numerically consistent, and complete constraint system, then S has a model.
4. If S contains a clash or is not numerically consistent, then S does not have a model.

Proof:

1. The termination proof is similar to the one for the tableau-based algorithm for $\mathcal{ALCN}$ [Donini *et al.*1991a]. In fact, termination is a consequence of the following properties of the completion rules.

   - As stated above, neither individual variables are substituted nor constraints are removed by the rules. The application of a completion rule strictly increases a constraint system.
   - All concepts occurring in S are subconcepts of C_0.
   - The maximal depth of constraints on an individual $x \in \tau_S$ is bounded by the maximal depth of C_0 minus the length of the (unique) role path from x_0 to x.

1. Intersection: If $x:(C_1 \sqcap C_2) \in S$ and $x : C_1 \notin S$ or $x : C_2 \notin S$

$S \to_{\sqcap} S \cup \{x : C_1, x : C_2\}$

2. Union: If $x:(C_1 \sqcup C_2) \in S$ and $x : C_1 \notin S$ and $x : C_2 \notin S$

$S \to_{\sqcup} S_1 = S \cup \{x : C_1\}$

$S \to_{\sqcup} S_2 = S \cup \{x : C_2\}$

3. Numerical Existential Quantification: If $x:(\downarrow\alpha.D) \in S$ and $x : D[\alpha/\alpha_x] \notin S$

$S \to_{\downarrow} S \cup \{x : D[\alpha/\alpha_x]\}$

4. New Objects
If $xRy \notin S$ for all $y \in \tau$ and $m > 0$, $k \geq 0$ are maximal such that $\{x:(\exists R.E_1), \dots, x:(\exists R.E_m), x:(\forall R.D_1), \dots, x:(\forall R.D_k)\} \subseteq S$ and Rules 1–3 cannot be applied to S, then for each n with $1 \leq n \leq m$ and for each n-Partition $P = \biguplus_{1 \leq i \leq n} P_i = \{1, \dots, m\}$ of m let S_P be defined as follows:

$$S \to_R S_P = S \cup \{xRy_i, \mid 1 \leq i \leq n\} \cup \{x:(\geq n\ R)\} \cup \{y_i : E_j \mid 1 \leq i \leq n, j \in P_i\} \cup \{y_i : D_j \mid 1 \leq i \leq n, 1 \leq j \leq k\}$$

where $y_i \in \tau$ are new variables (i.e., variables not occurring in S).

5. Prophylactic new objects
If $xRy \notin S$ for all $y \in \tau$ and $x:(=\ 0\ R) \notin S$ and k maximal with $x:(\forall R.D_i) \in S$ for $1 \leq i \leq k$, $x:(\mathsf{rel}\, N\ R) \in S$ for $N \in \mathbb{N}$ or $N = \alpha_y$ for some $y \in \tau, \alpha \in N_V$ and Rules 1–4 cannot be applied to S, then S_1, S_2 are defined as follows:

$S \to_n S_1 = S \cup \{x:(=0\ R)\}$

$S \to_n S_2 = S \cup \{xRy\} \cup \{y : D_i \mid 1 \leq i \leq k\} \cup \{x:(>0\ R)\}$

where $y \in \tau$ is a new variables (i.e., a variable not occurring in S).

Figure 6.10: The completion rules for $\mathcal{ALCUEN}^S$.

- Finally, Rules 4 and 5 are applied at most once to each individual and each role name, and they introduce only finitely many direct role successors for each individual.

2. We only consider Rules 3, 4 and 5 since Rules 1 and 2 are obvious. If S' is generated by the application of a completion rule to S, then $S \subseteq S'$, hence each model of S' is clearly a model of S. Thus we only have to

consider the other direction.

Numerical Existential Quantification: Application of this rule adds a constraint $x\!:\!C[\alpha/\alpha_x]$ to S, if $x\!:\!\downarrow\!\alpha.C$ is contained in S. If $\mathcal{I},\pi,\nu$ satisfy S, then we know that there exists an $n \in \mathbb{N}$ such that $\pi(x) \in \nu(C[\alpha/n])^{\mathcal{I}}$. Since the variable α_x does not occur in S (by our assumption that every variable is bound only once in the input concept), we can assume without loss of generality that $\nu(\alpha_x) = n$, and thus $\mathcal{I},\pi,\nu$ satisfy $x\!:\!C[\alpha/\alpha_x]$.

New Objects: Let x, R, k, m be as specified in the precondition of Rule 5 and let $\mathcal{I}$ satisfy S. Then there exist some $\ell \leq m$ and $z_1, \dots, z_\ell \in \Delta^{\mathcal{I}}$ such that

- $(\pi(x), z_i) \in R^{\mathcal{I}}$ for all i with $1 \leq i \leq \ell$,
- for all $1 \leq j \leq m$ there is some $j' \in \{1, \dots, \ell\}$ with $z_{j'} \in {E_j}^{\mathcal{I}}$, and
- for all $1 \leq j \leq k$, for all $1 \leq i \leq \ell$ it holds that $z_i \in {D_j}^{\mathcal{I}}$.

Since m, k are maximal and Rules 1-3 cannot be applied to S, the constraints mentioned above on R-successors of x are the only ones implied by S. Let P be the ℓ-partition of m with $j \in P_{j'}$ iff $z_{j'} \in {E_j}^{\mathcal{I}}$. In the corresponding constraint system S_P, ℓ new variables y_i are introduced. Let $\pi(y_i) = z_i$ for $1 \leq i \leq \ell$. Then

- $(\pi(x), \pi(y_i)) \in R^{\mathcal{I}}$ for all $1 \leq i \leq \ell$,
- for each $x\!:\!(\exists R.E_j) \in S$ holds $(\pi(x), \pi(y_{j'})) \in R^{\mathcal{I}}$ and $\pi(y_{j'}) \in {E_j}^{\mathcal{I}}$ for $j \in P_{j'}$,
- $\pi(y_i) \in {D_j}^{\mathcal{I}}$ for all $1 \leq i \leq \ell$, $1 \leq j \leq k$, and
- ${x_R}^{\mathcal{I}} \geq \ell$.

Hence $\mathcal{I}$ satisfies S_P.

Prophylactic New Objects: Let x, R, k be as specified in the precondition of rule 5 and let $\mathcal{I}$ satisfy S. Two cases are to be distinguished: If ${x_R}^{\mathcal{I}} = 0$, then clearly $\mathcal{I}$ satisfies S_1. Now let ${x_R}^{\mathcal{I}} > 0$ with $(\pi(x), z) \in R^{\mathcal{I}}$ for some $z \in \Delta^{\mathcal{I}}$. If we define $\pi(y) = z$, then $\mathcal{I}$ satisfies $S_2 = S \cup \{xRy\} \cup \{x_R > 0\} \cup \{y\!:\!D_i \mid 1 \leq i \leq k\}$.

3. As usual, we construct the canonical interpretation $\mathcal{I}_S$ induced by S: $\Delta^{\mathcal{I}_S}$ consists of the individual variables occurring in S, $(x, y) \in R^{\mathcal{I}_S}$ iff $xRy \in S$, and $x \in A^{\mathcal{I}_S}$ iff $x : A \in S$. This yields a tree-like interpretation, which needs not to be a model of S, since some number restrictions might not be satisfied for the following reasons: Either (a) an individual does not have any role successors, but their existence is implied by number

restrictions, or (b) it has some, but not sufficiently many role successors. Note that exact numerical restrictions on the number of role successors are given by a solution in $(\mathbb{N}, <)$ of the numerical constraints (which are satisfiable since S is numerically consistent). In the first case, S does not contain any constraints on such role successors, and we can simply generate an appropriate number of them. In the second case, the idea is to add sufficiently many *copies* of some already existing role successor y together with y's role successors. Proceeding like this from the leaves to the root, we end up with a model of S. More precisely, we start with an interpretation $\mathcal{I}_0$ defined as follows:

$$\begin{array}{lcl} \Delta^{\mathcal{I}_0} & := & \tau_S, \\ R^{\mathcal{I}_0} & := & \{(x,y) \in \Delta^{\mathcal{I}_0} \times \Delta^{\mathcal{I}_0} \mid xRy \in S\} \ \text{ for role names } R, \\ A^{\mathcal{I}_0} & := & \{x \in \Delta^{\mathcal{I}_0} \mid x : A \in S\} \ \text{ for concept names } A. \end{array}$$

Since S is clash-free and complete, $\mathcal{I}_0$ obviously satisfies all constraints beside at-least number restrictions: This can be shown by induction on the structure of concepts similar to the proof of Part 2 of Lemma 6.10. Let $\alpha_x^{\mathcal{I}_0}, \alpha_y^{\mathcal{I}_0}, \ldots$ and $x_R^{\mathcal{I}_0}, y_R^{\mathcal{I}_0}, \ldots$ be a solution of S_{num}. A model of S is then recursively defined as follows:

(a) Let

$$m_x^R := \max\{n \mid x :(\geq nR) \in S\} \cup \{\alpha_y^{\mathcal{I}_0} \mid x :(\geq \alpha_y R) \in S\}.$$

If ${x_R}^{I_j} = 0$ and $m_x^R \geq 1$, then $\mathcal{I}_{j+1}$ is obtained from I_j by adding m_x^R new individuals a_{xRi} with $(x, a_{xRi}) \in R^{\mathcal{I}_{j+1}}$ for $1 \leq i \leq m_x^R$.

(b) If $x \in \tau_S$ and $\mathcal{I}_j$ satisfies all number restrictions on all role successors of x, ${x_R}^{I_j} = \ell > 0$, and $m_x^R = \ell + k$ for some $k \geq 1$ and m_x^R as defined above, then we choose an R-successor y of x. First we make k copies of y and all its role-successors. Then the copies of y are called a_{xRi}, and $\mathcal{I}_{j+1}$ is obtained from I_j by adding all copies to $\Delta^{\mathcal{I}_j}$ and by defining

$$R^{\mathcal{I}_{j+1}} := R^{\mathcal{I}_j} \cup \bigcup_{1 \leq i \leq k} (x, a_{xRi}).$$

If neither (a) nor (b) applies to $\mathcal{I}_n$, then $\mathcal{I}_n$ is obviously a model of S: All constraints that are not at-least number restrictions are already satisfied by $\mathcal{I}_0$, and they are still satisfied after the execution of Step (a) because S is complete and thus does not impose any constraints on the individuals added by Step (a) (otherwise, Rule 5 could be applied to S). They are also still satisfied after the execution of Step (b): The individuals added by Step (b) are copies of already existing R-successors which already satisfy all constraints that are imposed by S on R-successors. Finally, $\mathcal{I}_n$ satisfies also all at-least number restrictions in S.

Step (b) is so simple because $\mathcal{ALUEN}^S$ has the tree-model property and because the usage of a solution for S_{num} makes sure that we do not introduce more role successors than allowed by at-most number restrictions.

4. If S contains a clash, then it is obviously unsatisfiable. If S is complete and $\mathcal{I}$ is a model of S, then we can deduce a solution for S_{num} in $(\mathbb{N}, >)$ as follows: We define

$$\begin{aligned} x_R &:= {x_R}^{\mathcal{I}}, \\ \alpha_x &:= \nu(\alpha, x). \end{aligned}$$

Since $\mathcal{I}$ satisfies all constraints in S, this obviously yields a solution for S_{num} and thus S is numerically consistent. ∎

Theorem 6.19 Satisfiability of $\mathcal{ALUEN}^S$-concepts is decidable.

Proof: Again, since Lemma 6.18 is the same as Lemma 4.8 (besides the different logics), the same arguments which were used on in Remark 4.9 to show that Lemma 4.8 implies decidability of satisfiability of $\mathcal{ALC}$-concepts now yield decidability of satisfiability of $\mathcal{ALUEN}^S$-concepts. ∎

Unfortunately, since $\mathcal{ALUEN}^S$ is not closed under negation, subsumption cannot be reduced to satisfiability. A closer look at the specific form of the concept $C_{\mathcal{D}}$ introduced in Figure 6.9 reveals that it can be written as $C_{\mathcal{D}} = D_1 \sqcap \neg D_2$ for two $\mathcal{ALUEN}^S$-concepts D_1, D_2: In fact, D_1 is the first conjunct of $C_{\mathcal{D}}$ and D_2 is the negation of the remainder of $C_{\mathcal{D}}$. Note that D_1 does not contain numerical variables. Furthermore, all numerical variables occurring in the remainder of $C_{\mathcal{D}}$ are universally quantified, which shows that D_2 contains only existential quantification of numerical variables. Since $D_1 \sqcap \neg D_2$ is unsatisfiable iff $D_1 \sqsubseteq D_2$, this implies:

Theorem 6.20 Subsumption of $\mathcal{ALUEN}^S$-concepts is undecidable.

6.3 Related work

Certain Modal Logics and Description Logics can be translated into first-order logic such that only two different variable names occur in the formulae obtained by this translation; see Remark 4.3. Thus, decidability of subsumption and other inference problems for these languages follows from the known decidability result for $\mathcal{L}_2$, i.e., first-order logic with two variables and without function symbols [Mortimer1975]. Recently, this decidability result has been extended to $\mathcal{C}_2$, i.e., first-order logic with 2 variables and counting quantifiers [Grädel *et al.*1997]. Independently, it has been proved in [Pacholski *et al.*1997] that satisfiability of $\mathcal{C}_2$ formulae can be decided in nondeterministic, doubly exponential time. As an immediate consequence, satisfiability and subsumption for $\mathcal{ALCN}(\sqcup, \sqcap, \neg, ^{-1})$, the extension of $\mathcal{ALC}$ by number restrictions with inversion and Boolean operators on roles, is still decidable. It should be noted, however, that expressing composition of roles in predicate logic requires more than two variables.

Using sophisticated techniques for translating Description Logic concepts into formulae of Propositional Dynamic Logics, it has been shown in [De Giacomo & Lenzerini1996] that deciding satisfiability and subsumption for a very expressive extension of $\mathcal{ALC}_{\mathsf{reg}}\mathcal{N}$ is ExpTime-complete. This extension allows for *qualifying* number restrictions on atomic and inverse roles. The number restrictions on inverse roles together with value restrictions involving the transitive closure of roles lead to the loss of the finite model property, as the following example shows:

$$(= 0\ f^{-1}.\top) \sqcap (= 1\ f.\top) \sqcap (\forall f^{+}.(= 1\ f.\top) \sqcap (= 1\ f^{-1}.\top))$$

It is obviously satisfiable (for example, 0 together with f interpreted as the successor function on the nonnegative integers is surely an instance of this concept), but it does not admit a finite model.

To our knowledge, the only constructor available in other logics that has some of the expressive power of (existential) symbolic number restrictions is infinite disjunction. As a consequence, the decidability result presented in this chapter are not subsumed and do not subsume other results.

Chapter 7

Transitive relations in Description Logics

As argued in Chapter 3, transitive relations play an important rôle in the representation of aggregated objects. In the following, three different ways of extending Description Logics by transitive roles are investigated with respect to their computational complexity. Finally, it is discussed in how far these approaches are suitable for the representation of aggregated objects and which of the demands made in Chapter 3 are satisfied by these approaches.

Three extensions of the concept language $\mathcal{ALC}$ by different kinds of transitivity are discussed in this section:

- **Transitive closure of roles:** In $\mathcal{ALC}_+$, the operator $^+$ can be applied to role names. The role R^+ is then interpreted as the *smallest* transitive relation containing R.
- **Transitive roles:** In $\mathcal{ALC}_{R^+}$, certain roles must be interpreted as transitive roles, but without the possibility to relate them to a "generating" role as in the first extension.
- **Transitive orbits of roles:** In $\mathcal{ALC}_\oplus$, the operator $^\oplus$ can be applied to role names. The role $R^\oplus$ is then interpreted as some (not necessarily the smallest) transitive relation containing R.

As a consequence of the results given in [Fischer & Ladner1979; Pratt1979], the basic inference problems for $\mathcal{ALC}_+$, i.e., $\mathcal{ALC}$ extended by the transitive closure of roles, are ExpTime-complete, whereas these problems are PSpace-complete for pure $\mathcal{ALC}$. Since one might not be able or willing to pay this high price for transitivity, looking for alternatives to the transitive closure is a quite natural step to undertake. Using results from Modal Logic [Ladner1977; Halpern & Moses1992], we show that these problems remain PSPACE-complete for $\mathcal{ALC}_{R^+}$. Finally, we prove that for $\mathcal{ALC}_\oplus$ these problems are as hard as for $\mathcal{ALC}_+$, namely EXPTIME-complete.

7.1 $\mathcal{ALC}$ extended by the Transitive Closure of Roles

In Section 4.5.1, syntax and semantics of $\mathcal{ALC}_+$ have already been introduced and in Section 4.5.3, the expressive power added by the transitive closure operator is described. We recall that $\mathcal{ALC}_+$ does no longer have the finite-tree-model property, but the finite-model and the tree-model property. As mentioned above and in Section 4.5.1, satisfiability and subsumption of $\mathcal{ALC}_+$-concepts are EXPTIME-complete. These problems are in PSPACE for $\mathcal{ALC}$, hence the source of this high computational complexity is the transitive closure operator.

This increase in computational complexity is due to the fact that, using universal value restrictions on transitive roles, one can propagate constraints on all role-chain successors. For example, the concept

$$C_0 := A \sqcap (\exists R.A) \sqcap (\forall R^+.\exists R.A)$$

makes sure that each individual reachable over an R-path of arbitrary but finite length has itself an R-successor. To prevent a tableau-based algorithm from generating all these R-successors (and thus looping), cycle detection mechanisms are necessary. In Section 4.5.1, these cycle detection mechanisms were already motivated. We recall that one has to check, for each individual that is generated by the algorithm, whether its constraints are the same as those on one of its role predecessor. If this is the case, one has furthermore to distinguish between cases where constraints on individuals propagated along some role chain are simply re-generated but satisfied and cases where the satisfaction of constraints is always postponed. Beside others, these cycle detection mechanisms require the storage of all constraints on all role predecessors of an individual whose constraints are tested for satisfiability. When started with the constraint system $\{x_0 : C_0\}$, the constraints on a role successor of x_0 can involve any subset of the subconcepts of C_0. Let n be the length of C_0. Then the number of subconcepts of C_0 is linear in n, but the number of subsets of subconcepts of C_0 is exponential in n. Intuitively, EXPTIME-hardness of

$\mathcal{ALC}_+$ is due to the fact that there are satisfiable $\mathcal{ALC}_+$-concepts where every model has a role chain of length exponential in the length of the input concept, that the storage of the constraints along this chain is necessary, and that this storage needs space exponential in the length of the input concept. For example, replacing $\oplus$ by $+$ in the $\mathcal{ALC}_\oplus$ -concept D defined in Section 7.3 yields an $\mathcal{ALC}_+$-concepts D' whose smallest models have an R-path whose length is exponential in the length of D'.

In Sections 7.2 and 7.3, it will be shown that the source of ExpTime-hardness is not transitivity of roles per se, but the interaction between an ordinary relation R and a transitive superrelation, here R^+.

7.2 $\mathcal{ALC}$ extended by Transitive Roles

When the correspondence between Modal Logics and Description Logics was discovered, results from the field of Modal Logic gave new insight into problems concerning Description Logics [De Giacomo & Lenzerini1994a; 1994b; Schild 1994; De Giacomo1995]: For example, it is well-known [Schild1991] that $\mathcal{ALC}$ is a notational variant of the propositional Multi-Modal Logic $\mathbf{K}_n$. Results for the Modal Logic $\mathbf{K4}_n$, which is a Multi-Modal Logic with n so-called agents extending propositional logic, gave the impetus to look closer at transitive roles as a different (and hopefully cheaper) way to extend Description Logics by transitive relations. The results will be given in this section.

If $\mathcal{ALC}$-interpretations are restricted to those where all role names are interpreted as transitive relations, then there is a 1-1 correspondence between $\mathbf{K4}_n$-formulae and $\mathcal{ALC}$-concepts such that a $\mathbf{K4}_n$-formula ϕ is satisfiable iff its translation into an $\mathcal{ALC}$-concept is satisfiable with respect to the restricted semantics. However, this restricted semantics is too restricted: As we have argued in Chapter 3, adequate representation of aggregated objects requires transitive relations, but this does not mean that *all* relations should be transitive. In this Section, we will investigate $\mathcal{ALC}$ extended with transitive roles—beside ordinary roles.

In [Halpern & Moses1992] it is shown that satisfiability of $\mathbf{K4}_n$ formulae is PSpace-complete. We will extend this result to mixed $\mathbf{K4}_\ell$+$\mathbf{K}_m$ formula, or, in Description Logics vocabulary, to $\mathcal{ALC}$ with transitive and ordinary roles. This extension is straightforward but not trivial: For example, the Modal Logic of one equivalence relation, $\mathbf{S5}_1$, is NP-complete, whereas the combination of two of these logics, $\mathbf{S5}_1$+$\mathbf{S5}_1$, is PSpace-complete.

Definition 7.1 $\mathcal{ALC}_{R^+}$ is an extension of $\mathcal{ALC}$ obtained by allowing the use of *transitive roles* inside concepts. The set of role names N_R is a disjoint union of role names $N_P = \{P_1, P_2, \dots\}$ and role names $N_+ = \{R_1, R_2, \dots\}$. In addition to what holds for $\mathcal{ALC}$-interpretations, an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ has to interpret role names $R_i \in N_+$ as transitive roles, i.e., an interpretation has to satisfy the additional condition

$$\text{if } (d,e) \in R_i^{\mathcal{I}} \text{ and } (e,f) \in R_i^{\mathcal{I}}, \text{ then } (d,f) \in R_i^{\mathcal{I}}$$

for each role $R_i \in N_+$.

In this section, a tableau-based algorithm is presented which tests for the satisfiability of $\mathcal{ALC}_{R^+}$-concepts. The algorithm extends and combines those presented in [Halpern & Moses1992] for Multi-Modal Logics in order to deal with the simultaneous use of both ordinary and transitive roles. It will be shown that this algorithm uses space polynomial in the length of the input concept.

The tableau-based algorithm given below is similar to the one given in Section 4.4—except for the modifications that were necessary for the correct handling of transitive roles, and the modifications that came in because this algorithm should use only polynomial space. One difference can be found in the nondeterminism of Rule 2 which handles disjunction. Another difference is that the role-successorship between two variables x, y is no longer represented by a constraint xRy, but by the fact that a node y—which contains all constraints concerning y—is an R-successor of the node x—which contains all constraints concerning x. The canonical model of a satisfiable concept is constructed by using the whole tree generated by the completion algorithm (instead of just using a clash-free leaf). Again, all concepts are supposed to be in negation normal form.

Definition 7.2 *Constraints* are of the form $x : C$ for an $\mathcal{ALC}_{R^+}$-concept C in NNF and a variable $x \in \tau$. When started with an $\mathcal{ALC}_{R^+}$-concept C_0, the completion algorithm starts with a tree consisting of a single node, the root x_0, labelled with the constraint system $\{x_0 : C_0\}$. Using the rules given in Figure 7.1, the completion algorithm expands this tree by adding either new nodes labelled with constraint systems or new constraints to labels of nodes. Such a tree, namely a tree where each node x is labelled with a set of constraints S_x and whose edges are labelled with role names in N_R, is called a *completion-tree*. A node y that is a successor of a node x is called a P-successor of x if the edge between them is labelled with P. For a transitive role $R \in N_+$, y is called an R-successor of x if there is a path from x to y where each edge is labelled with R. A node x is called an ancestor of a node y if there is a path from x to y regardless of the labels of its edges. The set of variables occurring in a completion tree $\mathcal{T}$ is the same as the set of its nodes and is denoted by $\tau_{\mathcal{T}}$. An interpretation $\mathcal{I}$ is a *model* of a completion-tree $\mathcal{T}$ iff there is a mapping $\pi : \tau_{\mathcal{T}} \to \Delta^{\mathcal{I}}$ that maps all nodes of $\mathcal{T}$ to individuals in $\Delta^{\mathcal{I}}$ such that

- $\mathcal{I}$ satisfies all constraints in all constraint systems that label nodes in $\mathcal{T}$, i.e., $\pi(x) \in C^{\mathcal{I}}$ for all nodes x and all constraints $x : C \in S_x$, and
- $(\pi(x), \pi(y)) \in R^{\mathcal{I}}$ for all $x, y \in \tau_{\mathcal{T}}$ where y is an R-successor of x.

The rules given in Figure 7.1 are applied only to leafs labelled with clash-free constraint systems. The completion algorithm terminates if no more rules can be applied. A tree is called *complete* if no more rules can be applied to it, and *clash-free* if all nodes are labelled with clash-free constraint systems. When started with a constraint system $\{x_0 : C_0\}$, the completion algorithm answers with "C_0 is satisfiable" if and only if the rules can be applied in such a way that they generate a complete and clash-free tree.

To present the rules in a unified way the following abbreviations are introduced:

Definition 7.3 Let $x, y \in \tau$, $P \in N_P$ and $R \in N_+$ be (transitive) role names, and let S_x be the label of a node in a completion tree. Then the x-consequences S_x/Py (resp. S_x/Ry) for y via P (resp. R) are defined as follows:

$$\begin{aligned} S_x/Py &:= \{y : C \mid x : \forall P.C \in S_x\} \\ S_x/Ry &:= \{y : C \mid x : \forall R.C \in S_x\} \cup \{y : \forall R.C \mid x : \forall R.C \in S_x\} \end{aligned}$$

Similar to Lemma 4.8, the proof of soundness and completeness can be split into 4 subtasks.

Lemma 7.4 Let C_0 be an $\mathcal{ALC}_{R^+}$-concept in NNF, let $\mathcal{T}$ be a tree obtained by applying the completion rules to $\{x_0 : C_0\}$. Then

1. For each completion rule $\mathcal{R}$ that can be applied to $\mathcal{T}$, and for each interpretation $\mathcal{I}$, (i) and (ii) are equivalent.
   (i) $\mathcal{I}$ is a model of $\mathcal{T}$.
   (ii) $\mathcal{R}$ can be applied in such a way that it yields some $\mathcal{T}'$ satisfied by $\mathcal{I}$.

1. Conjunction: If $x:(C_1 \sqcap C_2) \in S_x$ and $x:C_1 \notin S_x$ or $x:C_2 \notin S_x$, then $S_x \rightarrow\ S_x \cup \{x:C_1, x:C_2\}$

2. Disjunction: If $x:(C_1 \sqcup C_2) \in S_x$ and $x:C_1 \notin S_x$ and $x:C_2 \notin S_x$, then $S_x \rightarrow\ S_x \cup \{x:D\}$ for $D \in \{C_1, C_2\}$

3. Value and existential restriction:

If Rule 1 and Rule 2 cannot be applied to S_x, then

- for each $x:\exists P.C \in S_x$ with $P \in N_P$ create a P-successor y labelled with

$$S_y := \{y:C\} \cup S_x/Py \text{ for a new } y \in \tau.$$

- for each $x:\exists R.C \in S_x$ with $R \in N_+$, if there is no ancestor y of x where $\{y:C\} \cup S_x/Ry \subseteq S_y$, then create an R-successor y labelled with

$$S_y := \{y:C\} \cup S_x/Ry \text{ for a new } y \in \tau.$$

Figure 7.1: The completion rules for $\mathcal{ALC}_{R^+}$.

2. If $\mathcal{T}$ is a complete and clash-free completion tree, then $\mathcal{T}$ has a model.
3. If $\mathcal{T}$ contains a clash, then $\mathcal{T}$ does not have a model.
4. The completion algorithm terminates when applied to $\{x_0 : C_0\}$.

Proof of Lemma 7.4.1: The "if" direction: All rules expand a completion tree, hence if $\mathcal{I}$ is a model of the tree $\mathcal{T}'$ obtained from $\mathcal{T}$ by applying $\mathcal{R}$, then it is clearly also a model of $\mathcal{T}$.

The "only if" direction: Let $\mathcal{I}$ be a model of $\mathcal{T}$. If Rule 1 can be applied to a leaf x, then $x:C_1 \sqcap C_2 \in S_x$ and $\mathcal{I}$ clearly satisfies $\mathcal{T}'$. If Rule 2 is applicable to a leaf x, then $x:C_1 \sqcup C_2 \in S_x$ and $\mathcal{I}$ satisfies either $x:C_1$ or $x:C_2$, hence Rule 2 can be applied in such a way that $\mathcal{I}$ is a model of the tree $\mathcal{T}'$ generated by Rule 2 from $\mathcal{T}$. Now, let Rule 3 be applicable to a leaf x in $\mathcal{T}$. It has to be shown that $\mathcal{I}$ satisfies all constraint systems S_y generated by Rule 3.

Let $\pi : \tau_{\mathcal{T}} \rightarrow \Delta^{\mathcal{I}}$ be the corresponding mapping of individual variables to elements of $\Delta^{\mathcal{I}}$, and let $S_y := \{y : C\} \cup S_x/Py$ be the label of one of the new nodes generated by Rule 3. Then $x:\exists P.C \in S_x$ and $\pi(x)$ has at least one

P-successor $b \in \Delta^{\mathcal{I}}$ with $b \in C^{\mathcal{I}}$ and $b \in D^{\mathcal{I}}$ for all D with $x : \forall P.D \in S_x$. Hence $\mathcal{I}$ with $\pi(y) = b$ clearly satisfies S_y.

Now, let $S_y := \{y : C\} \cup S_x/Ry$ for some $R \in N_+$ be the label of one of the new nodes generated by Rule 3. Then $x : \exists R.C \in S_x$ and $\pi(x)$ has at least one R-successor $c \in \Delta^{\mathcal{I}}$ with $c \in C^{\mathcal{I}}$. Since $\mathcal{I}$ satisfies $\mathcal{T}$, all constraints of the form $x : \forall R.D$ are also satisfied, and $c \in D^{\mathcal{I}}$ for all D with $x : \forall R.D \in S_x$. Finally, $R^{\mathcal{I}}$ is a transitive relation. Hence if c has an R-successor d, this individual is also an R-successor of $\pi(x)$, and thus $d \in D^{\mathcal{I}}$ for all D with $x : \forall R.D \in S_x$. Hence $\mathcal{I}$ also satisfies all constraints of the form $y : \forall R.D \in S_y$ for $x : \forall R.D \in S_x$, and thus satisfies the whole constraint system S_y.

As a consequence, $\mathcal{I}$ satisfies each constraint system that label a node added by Rule 3. Since the label S_y of a new node y contains only constraints concerning y, namely those of the form $y : C$, the extension of the mapping π to y is independent from the extension of π to other nodes y', and thus $\mathcal{I}$ satisfies all constraint systems labelling nodes added by Rule 3. Finally, Rule 3 does not change the labels of nodes in $\mathcal{T}$. Summing up, we have that $\mathcal{I}$ satisfies the tree $\mathcal{T}'$ generated by Rule 3. ■

Proof of Lemma 7.4.2: Let $\mathcal{T}$ be a complete and clash-free completion tree with variables $\tau_{\mathcal{T}}$, and let r^+ be the transitive closure of a binary relation r, i.e.,

$$r^+ := \bigcup_{n \in \mathbb{N},\, n \geq 1} r^n.$$

We define a model $\mathcal{I}$ as follows: For concept names A and role names $P \in N_P$ and $R \in N_+$ we define

$$\begin{aligned}
\Delta^{\mathcal{I}} &:= \tau_{\mathcal{T}}, \\
A^{\mathcal{I}} &:= \{x \in \Delta^{\mathcal{I}} \mid x \text{ is a node in } \mathcal{T} \text{ and } x : A \in S_x\}, \\
P^{\mathcal{I}} &:= \{(x, y) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid y \text{ is an } P\text{-successor of } x \text{ in } \mathcal{T}\}, \\
R^{\mathcal{I}} &:= \big(\{(x, y) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid y \text{ is an } R\text{-successor of } x \text{ in } \mathcal{T}\} \cup \\
&\qquad \{(x, y) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid x : \exists R.C \in S_x \text{ for some } C \text{ and } y \text{ is an an-} \\
&\qquad\qquad \text{cestor of } x \text{ in } \mathcal{T} \text{ with } \{y : C\} \cup S_x/Ry \subseteq S_y\}\big)^+.
\end{aligned}$$

$\mathcal{T}$ is a completion tree. Hence all constraint systems labelling nodes in $\mathcal{I}$ are on different variables. It remains to show by induction on the structure of concepts that $\mathcal{I}$ is indeed a model of $\mathcal{T}$:

- First, by definition, $\mathcal{I}$ satisfies all constraints of the form $x : A$ in $\mathcal{T}$ for concept names A.

- $\mathcal{T}$ is clash-free, hence $\mathcal{I}$ satisfies by definition all constraints of the form $x : \neg A$.

- Since $\mathcal{T}$ is complete, it follows by induction on the structure of concepts that constraints of the form $x : C_1 \sqcap C_2$, $x : C_1 \sqcup C_2$ are also satisfied.

- Next, $\mathcal{I}$ obviously interprets all P-successors in $\mathcal{T}$ as $P^{\mathcal{I}}$-successors in $\mathcal{I}$. Transitive role names $R \in N_+$ are interpreted by definition by transitive relations, i.e., by the transitive closure of a binary relation.

- For role names $P \in N_P$, it follows immediately from the completeness of $\mathcal{T}$ and by induction that $\mathcal{I}$ satisfies all constraints of the form $x : \exists P.C$.

- Furthermore, value restrictions of the form $x : \forall P.D$ are taken into account by S/Py, and since neither Rule 1 nor Rule 2 can be applied when a P-successor y of x is generated by Rule 3, all value restrictions concerning P-successors of x are taken care of.

- We now consider existential value restrictions on transitive roles. If an R-successor y was generated for some $x : \exists R.C$, then $x : \exists R.C$ is surely satisfied by $\mathcal{I}$. If no R-successor was generated for $x : \exists R.C$, then there exists an ancestor y of x with $\{y : C\} \cup S_x/Ry \subseteq S_y$. According to the definition of $R^{\mathcal{I}}$, $(x, y) \in R^{\mathcal{I}}$ for such an R-successor. In both cases, $\mathcal{I}$ satisfies constraints of the form $x : \exists R.C$ by induction.

- Finally, suppose there exists some $x \in \tau_{\mathcal{T}}$ such that $\mathcal{I}$ does not satisfy a constraint $x : \forall R.D \in S$. This is only possible if there exists some $y \in \Delta^{\mathcal{I}}$ with $(x, y) \in R^{\mathcal{I}}$ and $y \notin D^{\mathcal{I}}$. According to the definition of $R^{\mathcal{I}}$, this y is either (1) an R-successor of x in $\mathcal{T}$ or (2) an ancestor of x with $\{y : C\} \cup S_x/Ry \subseteq S_y$, or (3) an R-successor of such an ancestor. In all three cases, due to the definition of Rule 3, $y : D \in S_y$. By induction, $y \in D^{\mathcal{I}}$ in contradiction to the assumption.

∎

Proof of Lemma 7.4.3: If $\mathcal{T}$ contains a clash, there is some leaf x in $\mathcal{T}$ with $\{x : A, x : \neg A\} \subseteq S_x$. Hence no interpretation can satisfy S_x, and thus $\mathcal{T}$ is unsatisfiable. ∎

Proof of Lemma 7.4.4: For a constraint system S_x, the maximum role depth of S_x, $\mathsf{depth}(S_x)$, is the maximum of nested $(\exists R.C), (\forall R.C)$ concepts of the concepts occurring in S_x, i.e.,

$$\mathsf{depth}(S) := \mathsf{max}\{\mathsf{depth}(C) \mid C \in S\},$$

where $\mathsf{depth}(C)$ is already defined in the proof of Part 4 of Lemma 6.10.

Furthermore, $\mathsf{sub}(C)$ denotes the set of subconcepts of C:

$$\begin{array}{rcl}
\mathsf{sub}(A) & := & \{A\} \text{ for all concept names } A \in N_C \\
\mathsf{sub}(\neg C) & := & \{\neg C\} \cup \mathsf{sub}(C) \\
\mathsf{sub}(C \sqcap D) & := & \{C \sqcap D\} \cup \mathsf{sub}(C) \cup \mathsf{sub}(D) \\
\mathsf{sub}(C \sqcup D) & := & \{C \sqcup D\} \cup \mathsf{sub}(C) \cup \mathsf{sub}(D) \\
\mathsf{sub}(\exists R.C) & := & \{\exists R.C\} \cup \mathsf{sub}(C) \\
\mathsf{sub}(\forall R.C) & := & \{\forall R.C\} \cup \mathsf{sub}(C)
\end{array}$$

We recall that the number of subconcepts of C_0 is linear in the length of C_0. In the following, we assume that $\mathsf{sub}(C_0)$ is linearly ordered and that $\{P_1, \dots, P_\ell, R_1, \dots, R_n\}$ is the set of role names occurring in C_0, where P_i is a role name in N_P and R_i a role name in N_+.

Now let $m = |\mathsf{sub}(C_0)|$. Then $\mathsf{depth}(S_x) \leq m$ for all nodes x in $\mathcal{T}$. Since all nodes in $\mathcal{T}$ are labelled with subsets of $\mathsf{sub}(C_0)$, $|S_x| \leq m$ for all nodes x.

Besides showing termination, an upper bound for the space needed by the algorithm will be given, hence the depth of the tree constructed is investigated more closely.

Fact 1: If y is a P_i-successor of x, then $\mathsf{depth}(S_y) < \mathsf{depth}(S_x)$. If y is an R_i-successor of x, $C \in S_y$ and C is not of the form $(\forall R_i.C_1)$, then $\mathsf{depth}(C) < \mathsf{depth}(S_x)$.

Fact 2: If z is a P_i- or an R_i-successor of y, y is a P_j- or an R_j-successor of x for $i \neq j$, then $\mathsf{depth}(S_z) < \mathsf{depth}(S_x)$.

Fact 3: The only way that the depth of the labels does not decrease is along some R_i-path. Let $x_0, \dots, x_k$ be nodes on such a path labelled with clash-free constraint systems S_{x_j} such that each x_{j+1} is an R_i-successor of x_j. Then each $S_{x_{j+1}}$ can be divided into three parts $S1_{j+1}, S2_{j+1}$ and $S3_{j+1}$: The first, $S1_{j+1}$, consists of $S_{x_j}/R_i x_{j+1}$. The second is $S2_{j+1} = \{x_{j+1} : C\}$ where $x_j : (\exists R_i.C)$ led to the creation of x_{j+1}. The third consists of all those constraints that were added to $S_{x_{j+1}}$ by Rule 1 or 2.

By definition, we have $S1_j \subseteq S1_{j+1}$ and $\#S2_j = 1$ for all $1 \leq j \leq k$. If $S2_{j+1} = \{x_{j+1} : C\}$, then for all ancestors w of x_{j+1} (thus for all x_ℓ with $\ell < j$), we have

$$(S1_{j+1} \cup S2_{j+1})[x_{j+1}/w] = S_{x_j}/Rw \cup \{w : C\} \not\subseteq S_w$$

by definition of Rule 3. There are at most m different choices for $S2_j$ and at most m different choices for $S1_j$ along this R_i-path. Rule 3 tests only whether $S1_{j+1} \cup S2_{j+1}$ is not contained in the labelling of an ancestor of x_j before creating a new node x_{j+1}. Hence the number of different choices for subsets $S3_{j+1}$ do not contribute to the length of this R_i-path and we thus have $k \leq m^2$.

Collecting these facts, we have that along one path in the completion tree, the depth of the constraints labelling nodes on this path may not decrease for at most m^2 consecutive nodes, and that the depth can decrease at most m times. Hence the length of paths in the completion tree is bounded by m^3. Furthermore, it is of bounded out-degree.[1] Hence its construction terminates.

■

As a consequence of the facts in the proof of Lemma 7.4, the completion algorithm yields not only a decision procedure for satisfiability of $\mathcal{ALC}_{R^+}$-concepts, but also a PSpace decision procedure.

Theorem 7.5 Satisfiability of $\mathcal{ALC}_{R^+}$-concepts is PSpace-complete.

Proof: As a consequence of results in [Savitch1970], for showing that a problem is in PSpace, it suffices to give a *nondeterministic* decision procedure that uses only polynomial space. The completion algorithm handles disjunction in a nondeterministic way: In contrast to other completion algorithms presented within this thesis which test both possibilities for constraints of the form $x : C \sqcup D$, it chooses nondeterministically between $x : C$ and $x : D$.

As stated in the proof of Lemma 4, the tree $\mathcal{T}$ constructed by the tableau construction algorithm for C_0 is of depth at most m^3 where $|\mathsf{sub}(C_0)| = m$. Once this algorithm has visited all successors of a node without detecting a clash, it can forget about the subtree below this node and reuse the space where it was memorised. Each S_x is a subset of $\mathsf{sub}(C_0)$, hence each S_x can be stored in m bits—given that we have a table that can be used to associate

[1] An upper bound for the outdegree is m.

binary m-vectors with subsets of $\mathsf{sub}(C_0)$. Such a table can be stored easily in m times the space that is needed to store C_0. There are less than m concepts of the form $(\exists R.C)$ in $\mathsf{sub}(C_0)$, hence there are less than m subtrees directly below a node x, and we can memorise in m bits which of them still have to be investigated.

Summing up, at each moment the algorithm is running, it has to store the following information for its actual node x at depth h:

- S_x,
- which of the subtrees below x still have to be investigated, and
- these two pieces of information for each of its h ancestors.

This information can be stored in $m + m + h(m + m) = (1 + h)2m$ bits. The above mentioned table can be stored in m^2 bits. Since $h \leq m^3$, the tableau construction algorithm needs at most $c + m^2 + 2m + 2m^4$ bits of storage for some constant c.

As a consequence, satisfiability of $\mathcal{ALC}_{R^+}$-concepts is in PSPACE. Finally, PSPACE-hardness follows from the facts that $\mathcal{ALC}_{R^+}$ is an extension of $\mathcal{ALC}$, and that satisfiability of $\mathcal{ALC}$ was shown to be PSPACE-complete in [Schmidt-Schauß & Smolka1991]. ∎

As we have seen, worst-case complexity of $\mathcal{ALC}_{R^+}$ is lower than the one of $\mathcal{ALC}_+$. The price in expressive power one has to pay for this lower complexity is illustrated by the following example: Suppose devices, signal-lines and pipe-connections are three disjoint concepts and we want to describe devices whose direct parts are either signal-lines or devices. Furthermore, these devices must have, as a possibly indirect part, a pipe-connection. In $\mathcal{ALC}_+$, these devices can be described by the following concept:

$$\begin{aligned}\texttt{V\_Device} \;:=\; & \texttt{Device} \sqcap \\ & (\forall \texttt{has\_part}.(\texttt{Device} \sqcup \texttt{Signal\_Line})) \sqcap \\ & (\exists \texttt{has\_part}^+.(\texttt{Pipe\_Connection})).\end{aligned}$$

These devices cannot be described if only a transitive role `has_some_part` is available without the possibility to relate this transitive role to a (possible non-transitive) subrole. In $\mathcal{ALC}_{R^+}$, we can only refer to parts without the possibility to distinguish between direct and indirect parts. Replacing in the above example $\texttt{has\_part}^+$ and `has_part` by `has_some_part` yields an unsatisfiable concept.

7.3 $\mathcal{ALC}$ extended by Transitive Orbits

The gap in expressive power and in computational complexity separating $\mathcal{ALC}_+$ and $\mathcal{ALC}_{R^+}$ motivated the search for a compromise between these two ways of extending Description Logics by transitive roles. As it turned out, there is one natural candidate for this compromise, which will be called *transitive orbits*. It allows one to relate a role to a transitive superrole, called the transitive orbit of the role. Since the relationship between a role and its transitive orbit is less strong than the one between a role and its transitive closure, one might hope that transitive orbits lead to a lower computational complexity than the transitive closure of roles.

Definition 7.6 $\mathcal{ALC}_\oplus$ is the extension of $\mathcal{ALC}$ obtained by allowing the use of *transitive orbits* of roles inside concepts. The transitive orbit of a role R is denoted $R^\oplus$ and interpreted as a transitive role containing $R^\mathcal{I}$, i.e., an $\mathcal{ALC}_\oplus$ -interpretation has to satisfy the additional conditions

$$R^\mathcal{I} \subseteq (R^\oplus)^\mathcal{I} \text{ and } (R^\oplus)^\mathcal{I} \text{ is transitive.}$$

A small example is given to highlight the difference between $\mathcal{ALC}_+$ and $\mathcal{ALC}_\oplus$ concepts. Let

$$\texttt{Carc\_Device} := \texttt{Device} \sqcap (\exists \texttt{has\_part}^\oplus.\texttt{Carcinogenic}),$$

and let $\mathcal{I}$ be a model of $\texttt{Carc\_Device}$. Then $d \in \texttt{Carc\_Device}^\mathcal{I}$ even if there is no $\texttt{has\_part}^\mathcal{I}$ chain from d to some $c \in \texttt{Carcinogenic}^\mathcal{I}$: For d being an instance of $\texttt{Carc\_Device}$ it is sufficient that there is some $c \in \texttt{Carcinogenic}^\mathcal{I}$ with $(d, c) \in (\texttt{has\_part}^\oplus)^\mathcal{I}$.

As a consequence, transitive orbits are easier to handle by tableau-based algorithms than the transitive closure of roles. When trying to build a model, value restrictions of the form $(\exists R^\oplus.C)$ can be treated by simply introducing an $R^\oplus$-successor y which does not need to be an R^n-successor for any n. Hence for y, beside C, we have to take care only of universal value restrictions on $R^\oplus$ and not of universal value restrictions on R. In contrast to this, for value restrictions of the form $(\exists R^+.C)$, in principle, we have to guess an n for that this value restriction is satisfied by some R^n-successor, and we have thus to take care also of universal value restrictions on R. For transitive orbits, it is neither necessary to guess such an n nor to take care of universal value restrictions on R.

In general, each model of an $\mathcal{ALC}_+$-concept D is also a model of its $\mathcal{ALC}_\oplus$-counterpart, which is obtained by replacing each R^+ in D by $R^\oplus$. If $C' \sqsubseteq D'$

holds for two $\mathcal{ALC}_\oplus$ concepts C', D', then clearly $C \sqsubseteq D$ holds for their $\mathcal{ALC}_+$-counterparts C, D. The converse does not hold:

$$(\forall R.(A \sqcap \neg A)) \sqsubseteq (\forall R^*.(A \sqcap \neg A))$$

holds if $*$ is replaced by $+$ (if x has no $R^\mathcal{I}$-successors, then it has clearly no $R^{+\mathcal{I}}$-successors), but it does not hold if $*$ is replaced by $\oplus$ (x can have an $R^{\oplus\mathcal{I}}$-successor without having an $R^\mathcal{I}$-successor).

Now the computational complexity of $\mathcal{ALC}_\oplus$ is investigated. The completion algorithm given in Section 7.2 can easily be modified to handle $\mathcal{ALC}_\oplus$ -concepts: It suffices to modify the definition of the x-consequences S_x/Ry (resp. $S_x/R^\oplus y$) for y via R (resp. $R^\oplus$)

$$\begin{aligned}
S_x/Ry &:= \{y : C \mid x : \forall R.C \in S\} \cup \\
& \{y : C \mid x : \forall R^\oplus.C \in S\} \cup \{y : \forall R^\oplus.C \mid x : \forall R^\oplus.C \in S\} \\
S_x/R^\oplus y &:= \{y : C \mid x : \forall R^\oplus.C \in S\} \cup \{y : \forall R^\oplus.C \mid x : \forall R^\oplus.C \in S\},
\end{aligned}$$

and to distinguish between R and $R^\oplus$-successors—where the first ones are generated for constraints of the form $x : \exists R.C$ and the latter ones for constraints of the form $x : \exists R^\oplus.C$. Furthermore, the test whether there already exists some predecessor labelled with a superset of the constraints to be generated has to be performed before creating any successor.

It can easily be seen that these modification yield soundness and completeness for $\mathcal{ALC}_\oplus$ -concepts. An important point to note is that R-successors generated by Rule 3 are labelled with consequences from value restrictions on R as well as on $R^\oplus$, whereas $R^\oplus$-successors are pure $R^\oplus$-successors. This is to say that their labels do not include any constraints $y : C$ for value restrictions $x : \forall R.C$. This is possible because an $R^\oplus$-successor does not need to be an R^n-successor for any n.

In contrast to the depth of the trees constructed for $\mathcal{ALC}_{R^+}$-concepts, the depth of trees constructed by this modified algorithm can no longer be bounded polynomially in the length of the concept. For example, if $\tilde{A}_i$ is defined as given below, each model of the concept

$$\begin{aligned}
D = {} & (\exists R.(\neg A_1 \sqcap \neg A_2 \sqcap \ldots \neg A_n)) \sqcap \\
& (\forall R^\oplus.((\exists R.\top) \sqcap (\tilde{A}_1 \sqcap \tilde{A}_2 \sqcap \ldots \sqcap \tilde{A}_n)))
\end{aligned}$$

has paths of length 2^n: The individuals on this path can be viewed as the representations of the binary encoding of the numbers from 0 to $2^n - 1$. Let $\mathcal{I}$ be an interpretation with $x \in D^\mathcal{I}$. The concept D is defined such that

- each $R^{\oplus}$-successor of x can be viewed as the representation of a number between 0 to $2^n - 1$,
- no $R^{\oplus}$-successor of x can represent both k and ℓ for $0 \leq k < \ell \leq 2^n - 1$,
- x has an R-successor representing 0, and
- for each of its $R^{\oplus}$-successors y, if y represents k, then it has an R-successor representing $k + 1 \mod 2^n$.

In order to ensure this, each $\tilde{A}_i$ expresses the correct switching of the i-th bit when counting from 0 up to $2^n - 1$. If y is an $R^{k+1^{\mathcal{I}}}$-successor of x, we have that $y \in A_i^{\mathcal{I}}$ iff the i-th bit in the binary encoding of k is equal to 1. More precisely,

$$\begin{aligned}
\tilde{A}_1 &= (A_1 \sqcap (\forall R.\neg A_1)) \sqcup (\neg A_1 \sqcap (\forall R.A_1)) \\
\tilde{A}_i &= (\bigsqcap_{1 \leq j < i} A_j \sqcap ((A_i \sqcap \forall R.\neg A_i) \sqcup (\neg A_i \sqcap \forall R.A_i)))\sqcup \\
&\quad (\neg \bigsqcap_{1 \leq j < i} A_j \sqcap ((A_i \sqcap \forall R.A_i) \sqcup (\neg A_i \sqcap \forall R.\neg A_i))).
\end{aligned}$$

The length of D is quadratic in n whereas each model $\mathcal{I}$ of D has an $R^{\mathcal{I}}$-path of length in $O(2^n)$.

Hence there are $\mathcal{ALC}_{\oplus}$ -concepts whose smallest models are of a relatively large depth, namely exponential in the size of the concept. This property is a good hint that satisfiability of $\mathcal{ALC}_{\oplus}$ -concepts cannot be decided using only polynomial space. As the following theorem shows, this is indeed true.

Theorem 7.7 Satisfiability of $\mathcal{ALC}_{\oplus}$-concepts is ExpTime-complete.

Proof: Satisfiability of $\mathcal{ALC}_{\oplus}$-concepts is in ExpTime because it can be decided by the modified tableau construction algorithm. It is easy to see that for an $\mathcal{ALC}_{\oplus}$-concept C_0, this modified algorithm creates a tree whose depth is exponentially bounded by the length of C_0 because a successor of a node x labelled with S_y is only generated if no ancestor is labelled with a superset of S_y.

To show that satisfiability of $\mathcal{ALC}_{\oplus}$-concepts is indeed ExpTime-hard, we can modify the proof ExpTime-hardness of the satisfiability of PDL formulae given in [Fischer & Ladner1979]. The proof gives, for an alternating Turing Machine M and an input word x, a PDL formula $f_{M(x)}$ such that $f_{M(x)}$ is satisfiable iff x is accepted by a simplified trace of M. A translation of PDL formulae into $\mathcal{ALC}_{+}$-concepts can be found in [Schild1991]. This translation is satisfiability preserving and yields a concept whose length is linear in the length of the input concept. If this translation is applied to $f_{M(x)}$, it yields an $\mathcal{ALC}_+$-concept $D_{M(x)}$ using a single role R and its transitive closure R^+. This concept is of the form

$$D_{M(x)} = C_1 \sqcap C_2 \sqcap (\forall R^+.C_2),$$

where R is the only role name occurring in $C_{M(x)}$. Furthermore, R^+ occurs neither in C_1 nor in C_2. Because of this special form, $D_{M(x)}$ is satisfiable iff its $\mathcal{ALC}_\oplus$ -counterpart $D'_{M(x)} = C_1 \sqcap C_2 \sqcap (\forall R^\oplus.C_2)$ is satisfiable:

Each model of $D_{M(x)}$ is clearly a model of $D'_{M(x)}$. Now, let $\mathcal{I}'$ be a model of $D'_{M(x)}$ with $x \in {D'_{M(x)}}^{\mathcal{I}'}$. Then $x \in C_1^{\mathcal{I}'} \sqcap C_2^{\mathcal{I}'}$ and for all y with $(x, y) \in R^{\oplus\mathcal{I}'}$ it holds that $y \in C_2^{\mathcal{I}'}$. Let $\mathcal{I}$ be an interpretation of $D_{M(x)}$ which is equal to $\mathcal{I}'$ for concept and role names in C_1, C_2 and where $R^{+\mathcal{I}}$ is the transitive closure of $R^{\mathcal{I}'}$. Hence we have that $(x, y) \in R^{+\mathcal{I}}$ implies $(x, y) \in R^{\oplus\mathcal{I}'}$ for all $x, y \in \Delta^{\mathcal{I}}$. It follows that $y \in C_2^{\mathcal{I}}$ for all $(x, y) \in R^{+\mathcal{I}}$, which yields $x \in D_{M(x)}^{\mathcal{I}}$. Hence $\mathcal{I}$ is a model of $D_{M(x)}$. ■

7.4 Representation of part-whole relations

In this section, the general observations concerning part-whole relations made in Chapter 3 are related to the results concerning expressive power and computational complexity of transitive relations in Description Logics in this chapter.

As discussed in Chapter 3, the minimal prerequisite for the adequate representation of part-whole relations is the possibility to express that a certain relation—the general part-whole relation $\prec$—is transitive. This possibility is given in all extensions of $\mathcal{ALC}$ investigated in this chapter. Hence, if only this general part-whole relation is needed, the "cheapest" kind of transitivity, namely transitive roles, is sufficient. If $\mathcal{ALC}$ is chosen as the basic Description Logic, this means that $\mathcal{ALC}_{R^+}$ is the appropriate extension for the application under consideration.

In case the application asks for the distinction between different part-whole relations, the question arises which Description Logic has enough expressive power for an adequate representation of these part-whole relations. At the same time, the according inference problems should be of a computational complexity that is as low as possible. We will first consider integral part-whole relations. Given an integrity condition $\mathsf{Int}(.)$, the according integral part-whole relation $\prec_{\mathsf{Int}}$ was defined in Section 3.1.2 as

$$x \prec_{\mathsf{Int}} y \overset{\text{def.}}{\iff} x \prec y \wedge \mathsf{Int}(x).$$

In $\mathcal{ALC}_{R^+}$, given a transitive role `part_of`, integral part-whole relations can be represented as follows: First, introduce a role name `intpw` and a concept `Int` for the integral part-whole relation $\prec_{\mathsf{Int}}$. Then the following substitutions will force all value restrictions to be interpreted in the appropriate way, namely to address exactly those parts which are integral with respect to `Int`.

$$\begin{array}{lcl}\exists\texttt{intpw}.C & \rightsquigarrow & \texttt{Int}\sqcap\exists\texttt{part\_of}.C\\ \forall\texttt{intpw}.C & \rightsquigarrow & \neg\texttt{Int}\sqcup\forall\texttt{part\_of}.C\end{array}$$

In this solution, the concept `Int` introduced for an integrity condition can be atomic or complex. Hence, concepts representing integrity conditions can be ordered with respect to the subsumption relation, which induces a natural hierarchy on the corresponding integral part-whole relations: For example, let $\prec'_{\mathsf{Int}}\subseteq\prec_{\mathsf{Int}}$. If the corresponding integrity conditions are modeled in an appropriate way, i.e., $\texttt{Int}'\sqsubseteq\texttt{Int}$, it follows immediately that $(\exists\texttt{intpw}'.C)\sqsubseteq(\exists\texttt{intpw}.C)$. Furthermore, integral part-whole relations inherit transitivity from the general part-whole relation. This is to say that

$$\forall\texttt{intpw}.C\sqsubseteq\forall\texttt{intpw}.(\forall\texttt{intpw}.C)$$

(which is a translation of the axiom for transitive frames in Modal Logic, see [Halpern & Moses1992]).

As a consequence, $\mathcal{ALC}_{R^+}$ is also expressive enough for the adequate representation of the general part-whole relation together with a hierarchically structured set of integral part-whole relations—provided that the integrity conditions can be expressed in $\mathcal{ALC}_{R^+}$. Hence $\mathcal{ALC}$ can be extended by the general part-whole relation together with an important class of part-whole relations without losing PSpace-completeness of the relevant inference problems.

In contrast to this, composed part-whole relations as subrelations of the general part-whole relation ask for more expressive power. In Section 3.1.3, these composed part-whole relations were defined as

$$x\prec_{\mathsf{Comp}}y\overset{\text{def.}}{\Longleftrightarrow}x\prec y\wedge x\ \mathsf{CompRel}\ y.$$

To represent a single composed part-whole relation $\prec_{\mathsf{Comp}}$, it suffices to use, as the general part-whole role, the transitive orbit $\texttt{CompRel}^{\oplus}$ of a role `CompRel` representing the additional condition CompRel.

If more than one composed part-whole relation must be represented, transitive orbits or the transitive closure are no longer sufficient because they allow only to relate *one* transitive role to *one* subrole. To overcome this problem, one could use $\mathcal{ALC}_{\mathsf{reg}}$ instead of $\mathcal{ALC}_{+}$ and represent the general part-whole relation as the transitive closure of the union of all its sub-part-whole relations. Alternatively, one could use $\mathcal{ALCH}_{R^+}$, a Description Logic introduced in [Horrocks & Gough1997], which was designed to overcome the above mentioned lack in expressive power: It extends $\mathcal{ALC}_{R^+}$ by *role-hierarchies*, that is the possibility to specify inclusion axioms $R \sqsubseteq S$ on roles—regardless of whether they are transitive or not. For example, to represent two non-transitive composed part-whole relations $\prec_{\mathsf{Comp}}, \prec'_{\mathsf{Comp}}$, one introduces two non-transitive role names $\mathtt{CompRel}, \mathtt{CompRel}'$ and one transitive role name $\mathtt{part\_of}$ and adds the inclusion axioms

$$\mathtt{CompRel} \sqsubseteq \mathtt{part\_of} \quad \text{and} \quad \mathtt{CompRel}' \sqsubseteq \mathtt{part\_of}.$$

A model of these axioms interprets both $\prec_{\mathsf{Comp}}$ and $\prec'_{\mathsf{Comp}}$ as subroles of the transitive role $\mathtt{part\_of}$. Although the extension of $\mathcal{ALC}$ by inclusion axioms on possibly transitive roles leads to ExpTime-hardness of the corresponding inference problems (since transitive orbits can be expressed), the authors of [Horrocks & Gough1997] claim that the algorithm behaves quite well in practice, that it is much simpler than the algorithms for $\mathcal{ALC}_+$, and that it is amenable to optimisation techniques.

Summing up, there are mainly two alternatives for the representation of part-whole relations in Description Logics:

1. If the application does not ask for the representation of composed part-whole relations, then $\mathcal{ALC}_{R^+}$ is expressive enough, even if hierarchically structured integral part-whole relations are needed. In this case, the inference problems are still in PSpace.

2. If the application asks for composed part-whole relations besides the general part-whole relation, then $\mathcal{ALC}_{\oplus}$ or $\mathcal{ALCH}_{R^+}$ are the least expressive propositionally closed Description Logic able to represent arbitrary roles together with transitive superroles. Unfortunately, even for $\mathcal{ALC}_{\oplus}$, the relevant inference problems are ExpTime-hard.

Chapter 8

Conclusion

This thesis reports the investigations that were made to find out which knowledge representation system based on Description Logics is best suited for supporting the design of process models. To be well-suited for the process engineering application, a Description Logic has to provide the expressive power to describe relevant properties of the building blocks process models consist of. Furthermore, this Description Logic has to support the structured storage of these building blocks. To satisfy the latter requirement, the relevant inference problems should be of an acceptable computational complexity, namely at least decidable. The goal of these investigations was not to implement a complete DL system, but to extend already existing Description Logics such that they provide the expressive power required for the process engineering application, and to investigate the computational properties of these Description Logics.

As a matter of fact, no such "best suited" DL system could be chosen. This is not only due to the limited amount of time available for these investigations, but also due to the fact that such a best suited DL system does not need to exist: Roughly spoken, the process engineering application asks for a DL system with rather high expressive power and decidable inference problems. Now, the expressive power of a DL system is proportional to the computational complexity of its relevant inference problems. Additionally, expressive power is a multi-dimensional property and thus there are several candidates for a best suited DL system. The most important of these candidates were defined and investigated with respect to the computational complexity of the relevant inference problems and expressive power.

To cope with the high complexity of process models, aggregated objects are widely used while building process models. As a consequence, the part-whole relation, which relates parts to aggregated objects, plays an important rôle for the description of (aggregated objects in) process models. The part-whole relation has some inherent properties which ask for an adequate representation, the most important of which is transitivity. Beside the general part-whole relation, it seems natural to use various sub-part-whole relations, whose meaning is mostly only given by intuitive examples in literature. For the adequate representation of the general part-whole relation and its sub-part-whole relations, exact definitions of their properties and the way in which they interact are necessary. Since these exact definitions were, to our knowledge, lacking, we established a taxonomy of part-whole relations which gives a precise scheme for the classification of sub-part-whole relations. This scheme implies both the main properties of the sub-part-whole relations defined within this scheme as well as the way in which they interact. With this taxonomy in mind, three ways of extending Description Logics by transitive relations were investigated, namely transitive roles, transitive orbits, and the transitive closure of roles.

Transitive roles per se, without referring to an underlying subrole, are algorithmically easier to handle than the transitive closure or transitive orbits of roles. Hence, when using a Description Logic system for an application that asks for the adequate representation of aggregated objects, it is worth asking which kind of sub-part-whole relations are important in this application: We could show that if, beside the general part-whole relation, only integral part-whole relations are needed, then transitive roles are expressive enough. However, the adequate representation of composed part-whole relations asks for transitive orbits or hierarchies of possibly transitive roles—which both increase the complexity of the relevant inference problems. All solutions enable us to define concepts or describe individuals by referring to parts at a level of decomposition not known and not bounded in advance. As a by-product, all solutions provide the expressive power to refer to ancestors, friends or relatives in any generation or in any degree of relationship.

The complexity results hold for the case where $\mathcal{ALC}$ is extended by transitive roles, the transitive closure, or the transitive orbits. An interesting question arising from these observations is whether these results hold for extensions of other Description Logics as well, but its investigation goes beyond the scope of this work.

Expressive number restrictions: Since a process model strongly depends on the way in which its building blocks are interconnected, the process engineering application asks for a Description Logic that has the expressive power to describe this structural information. Expressive number restrictions can provide this expressive power. This can be seen in the fact that, for example, $\mathcal{ALCN}(\circ)$ does not have the tree model property. Within this thesis, two approaches to make number restrictions more expressive were investigated.

Symbolic number restrictions turned out to be very expressive. For example, they allow to describe symmetry conditions or the dependency between the number of an objects connections and the number of the connections of its parts. Unfortunately, this high amount of expressive power makes all relevant inference problems undecidable. However, a slight restriction of this logic could be shown to have a decidable satisfiability problem—even though the according subsumption problem is still undecidable.

Number restrictions on complex roles proved also to be very expressive, but, in contrast to symbolic number restrictions, it was possible to design a new, expressive, and decidable Description Logic, namely $\mathcal{ALCN}(\circ)$. This does not hold for other extensions by either more complex roles inside number restrictions (than those built using composition only) or by a more expressive underlying Description Logic, namely $\mathcal{ALC}_+$. Within this thesis, a variety of open problems could be solved concerning the effects complex roles inside number restrictions have on the computational complexity of the relevant inference problems.

In order to evaluate the suitability of DL system as a means to structure the storage of process model building blocks, the DL system CRACK was integrated into the process modeling tool MODKIT. This integration is realized in such a way that browsing the class hierarchy, defining new classes, and migrating modified prototypes are supported by the system services provided by CRACK. The fact that the expressive power of CRACK is high enough for providing this support is due to the fact that all information concerning objects and classes in MODKIT that can be deduced automatically from their specifications can be expressed in CRACK. A process modeling tool where more of this information could be deduced could profit from the integration of a more expressive DL system.

8.1 Consequences of the complexity results for the process modeling application

The results of the investigations of the computational complexity of Description Logics can be summarised as follows: (a) Complex roles inside number restrictions lead to the undecidability of the relevant inference problems for some combinations of the role-forming constructors that include composition, whereas the basic Description Logic $\mathcal{ALC}$ extended by composition inside number restrictions still has decidable inference problems; for an overview see Figure 6.3. (b) If $\mathcal{ALC}$ is extended by symbolic number restrictions of the form motivated by the examples, it still has a decidable satisfiability problem. Unfortunately, subsumption of this extension by symbolic number restrictions turned out to be undecidable. (c) Transitive relations—which are needed for the adequate representation of aggregated objects—can be added to $\mathcal{ALC}$ without strongly increasing the computational complexity of the relevant inference problems. In contrast, different ways of using a transitive superrole together with a possibly non-transitive subrole strongly increasing this complexity.

For the representation of process model building blocks, this means that we have gained a better insight into "difficult" properties (those that necessitate constructors which lead to a dramatical increase of the computational complexity) of building blocks we would like to describe and "well-behaved" ones (those that can be expressed using constructors which lead to an acceptable computational complexity). Two examples for difficult properties and examples of well-behaved ones will follow.

If we define a class by restricting the number of objects related to its instances via complex roles involving conjunction and composition of roles, such as, for example, in the concept $(\geq 4\ ((\texttt{connected-to} \circ \texttt{has-phenomena}) \sqcap \texttt{has-phenomena}))$, then either the computation of the taxonomy runs into an infinite loop, or this information is simply ignored. Hence this property is an example for a difficult property. A second example for difficult properties are symmetry properties that must hold for instances of a specific class which can only be expressed using symbolic number restrictions. Then this information concerning symmetry can be taken into account for testing whether this class can ever be instantiated, but it has to be ignored for the computation of the taxonomy. This shortcoming of the inference services of a DL system are not due to an unwilling software developer or a slow computer, but they were proved to be a consequence of the theory of computation.

It turned out that properties of aggregated objects that are described by referring to its parts can be expressed using a Description Logic for which the complexity of the relevant inference problems is rather low—provided that we are not too demanding with respect to the different kinds of part-whole relations we would like to use to refer to these parts. Hence, these properties belong to the well-behaved ones, even though their description may refer to parts that are in a level of decomposition neither known nor bounded in advance.

Appendix A: Syntax rules

Complex concepts and possibly complex roles of the different concept languages referred to in this paper are built according to the following syntax rules where C, C_1, C_2 denote concepts, A stands for a concept name or a number restriction, R, R_1, R_2 denote roles, n stands for a nonnegative integer and α is a numerical variable.

$\mathcal{ALCN}$:

$$\begin{array}{rcl} C \longrightarrow & C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \neg C_1 \mid \\ & (\forall R.C_1) \mid (\exists R.C_1) \mid (\geq n\ R) \mid (\leq n\ R) \end{array}$$

$\mathcal{ALC}_+$:

$$\begin{array}{rcl} C \longrightarrow & C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \neg C_1 \mid \\ & (\forall R.C_1) \mid (\exists R.C_1) \\ R \longrightarrow & R \mid R_1^+ \end{array}$$

$\mathcal{ALC}_{\mathsf{reg}}$:

$$\begin{array}{rcl} C \longrightarrow & C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \neg C_1 \mid \\ & (\forall R.C_1) \mid (\exists R.C_1) \\ R \longrightarrow & R_1 \sqcup R_2 \mid R_1 \circ R_2 \mid R_1^+ \end{array}$$

$\mathcal{ALCN}^S$:

$$\begin{array}{rcl} C \longrightarrow & C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \neg C_1 \mid \\ & (\forall R.C_1) \mid (\exists R.C_1) \mid ({\downarrow}\alpha.C_1) \mid \\ & (\geq n\ R) \mid (\leq n\ R) \mid (\geq \alpha\ R) \mid (\leq \alpha\ R) \end{array}$$

$\mathcal{ALUEN}^S$:

$$\begin{array}{rcl} C \longrightarrow & C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \neg A \mid \\ & (\forall R.C_1) \mid (\exists R.C_1) \mid ({\downarrow}\alpha.C_1) \mid \\ & (\geq n\ R) \mid (\leq n\ R) \mid (\geq \alpha\ R) \mid (\leq \alpha\ R) \end{array}$$

$\mathcal{ALCN}(M)$:

$$\begin{array}{rcl} C \longrightarrow & C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \neg C_1 \mid \\ & (\forall R.C_1) \mid (\exists R.C_1) \mid \qquad (R \text{ is a role name }) \\ & (\geq n\ R') \mid (\leq n\ R') \mid (R' \text{ is built according to } M) \end{array}$$

$\mathcal{ALC}_+\mathcal{N}(M)$:

$$\begin{array}{rcl} C \longrightarrow & C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \neg C_1 \mid \\ & (\forall R.C_1) \mid (\exists R.C_1) \mid \qquad (R \text{ is a regular role}) \\ & (\geq n\ R') \mid (\leq n\ R') \mid (R' \text{ is built according to } M) \end{array}$$

Semantics

The following table contains a short description of the semantics of the most important role- and concept forming constructors for an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$. The letter n stands for a nonnegative integer.

Syntax	Semantics
Role names R	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
Concept names C	$C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
$\top$	$\Delta^{\mathcal{I}}$
$\bot$	$\emptyset$
$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
$\exists R.C$	$\{d \in \Delta^{\mathcal{I}} \mid \text{For some } e \in \Delta^{\mathcal{I}},\ (d,e) \in R^{\mathcal{I}} \text{ and } e \in C^{\mathcal{I}}\}$
$\forall R.C$	$\{d \in \Delta^{\mathcal{I}} \mid \text{For all } e \in \Delta^{\mathcal{I}} \text{ if, } (d,e) \in R^{\mathcal{I}} \text{ then } e \in C^{\mathcal{I}}\}$
$(\geq n\ R)$	$\{d \in \Delta^{\mathcal{I}} \mid \text{There are at least } n \text{ elements } e \in \Delta^{\mathcal{I}} \text{ with } (d,e) \in R^{\mathcal{I}}\}$
$(\leq n\ R)$	$\{d \in \Delta^{\mathcal{I}} \mid \text{There are at most } n \text{ elements } e \in \Delta^{\mathcal{I}} \text{ with } (d,e) \in R^{\mathcal{I}}\}$
$(R_1 \circ R_2)^{\mathcal{I}}$	$R_1^{\mathcal{I}} \circ R_2^{\mathcal{I}} = \{(d,f) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid \text{There exists } e \in \Delta^{\mathcal{I}} \text{ with } (d,e) \in R_1^{\mathcal{I}} \text{ and } (e,f) \in R_2^{\mathcal{I}}\}$
$(R^+)^{\mathcal{I}}$	$\cup_{i \geq 1} (R^{\mathcal{I}})^i, \text{ where } (R^{\mathcal{I}})^n = \underbrace{(R^{\mathcal{I}} \circ R^{\mathcal{I}} \circ \ldots \circ R^{\mathcal{I}})}_{n \text{ times}}$

Bibliography

[Artale *et al.* 1994] A. Artale, F. Cesarini, E. Grazzini, F. Pippolini, and G. Soda. Modelling composition in a terminological language environment. In *Workshop Notes of the ECAI Workshop on Parts and Wholes: Conceptual Part-Whole Relations and Formal Mereology*, pages 93–101, Amsterdam, 1994.

[Baader *et al.* 1993] F. Baader, M. Buchheit, and B. Hollunder. Cardinality restrictions on concepts. Technical Report RR-93-48, Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), Kaiserslautern, Germany, 1993.

[Baader *et al.* 1994] F. Baader, E. Franconi, B. Hollunder, B. Nebel, and H. Profitlich. An empirical analysis of optimization techniques for terminological representation systems, or: Making KRIS get a move on. *Applied Artificial Intelligence*, 4:109–132, 1994.

[Baader 1990a] F. Baader. Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles. Technical Report RR-90-13, Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), Kaiserslautern, Germany, 1990. An abridged version appeared in *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence IJCAI-91*, pp. 446–451.

[Baader 1990b] F. Baader. A formal definition for expressive power of knowledge representation languages. In *Proceedings of the 9th European Conference on Artificial Intelligence, ECAI-90*, pages 53–58, Stockholm (Schweden), 1990.

[Baader 1991] F. Baader. Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, 1991.

[Baader & Hanschke 1991] F. Baader and P. Hanschke. A schema for integrating concrete domains into concept languages. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 452–457, Sydney, 1991.

[Baader & Hanschke 1993] F. Baader and P. Hanschke. Extensions of concept languages for a mechanical engineering application. In *Proc. of the 16th German AI-Conference, GWAI-92*, volume 671 of *Lecture Notes in Computer Science*, pages 132–143, Bonn, Germany, 1993. Springer-Verlag.

[Baader & Sattler 1996a] F. Baader and U. Sattler. Description logics with symbolic number restrictions. In W. Wahlster, editor, *Proceedings of the Twelvth European Conference on Artificial Intelligence (ECAI-96)*. John Wiley & Sons Ltd, 1996.

[Baader & Sattler 1996b] F. Baader and U. Sattler. Number restrictions on complex roles in description logics. Technical Report 96-02, LuFg Theoretical Computer Science, RWTH Aachen, 1996. Available via www: `http://www-lti.informatik.rwth-aachen.de/Forschung/Papers.html`.

[Baader & Sattler 1996c] F. Baader and U. Sattler. Number restrictions on complex roles in description logics. In *Proceedings of the Fifth International Conference on the Principles of Knowledge Representation and Reasoning (KR-96)*. Morgan Kaufmann, Los Altos, 1996.

[Baumeister *et al.* 1998] M. Baumeister, R. Bogusch, B. Lohmann, U. Sattler, and D. Souza. The Chemical Engineering Data Model Veda. Technical report, Lehrstuhl für Prozeßtechnik RWTH Aachen, 1998. To appear.

[Baumeister 1998] M. Baumeister. *Ein flexibles und abstrahierendes Objektmodell für die Modellierung chemischer Prozesse.* PhD thesis, Lehrstuhl für Prozeßtechnik, Lehrstuhl für Informatik V, RWTH Aachen, 1998. To appear.

[Beeri *et al.* 1997] C. Beeri, A. Y. Levy, and M.-C. Rousset. Rewriting queries using views in description logics. In *Proceedings of the Sixteenth ACM SIGACT SIGMOD Symposium on Principles of Database Systems (PODS-97)*, Tucson, Arizona,, May 1997.

[Bergamaschi & Sartori 1992] S. Bergamaschi and C. Sartori. On taxonomic reasoning in conceptual design. *ACM Transactions on Database Systems*, 17(3):385–422, 1992.

[Berger 1966] R. Berger. The undecidability of the dominoe problem. *Memoirs of the American Mathematical Society*, 66, 1966.

[Bogusch *et al.* 1996] R. Bogusch, B. Lohmann, and W. Marquardt. Computer-aided process modeling with MODKIT. In *Proceedings of Chemputers Europe III*, pages 29–30, Frankfurt, Germany, October 1996.

[Bogusch 1998] R. Bogusch. *Repräsentation mathematischer Modelle für die rechnergestützte Modellierung verfahrenstechnischer Prozesse.* PhD thesis, Lehrstuhl für Prozeßtechnik, RWTH Aachen, 1998. To appear.

[Borgida & Patel-Schneider 1994] A. Borgida and P. Patel-Schneider. A semantics and complete algorithm for subsumption in the CLASSIC description logic. *Journal of Artificial Intelligence Research*, 1:277–308, 1994.

[Brachman *et al.* 1991] R. J. Brachman, D. McGuinness, P. Patel-Schneider, L. Resnick, and A. Borgida. Living with CLASSIC: When and how to use a KL-ONE-like language. In J. F. Sowa, editor, *Principles of Semantic Networks*. Morgan Kaufmann, Los Altos, 1991.

[Brachman 1983] R. J. Brachman. What IS-A is and isn't: An analysis of taxonomic links in semantic networks. *IEEE Computer*, 16(10):30–36, October 1983.

[Brachman & Schmolze 1985] R. J. Brachman and J. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.

[Bresciani *et al.* 1995] P. Bresciani, E. Franconi, and S. Tessaris. Implementing and testing expressive description logics: A preliminary report. In A. Borgida, M. Lenzerini, D. Nardi, and B. Nebel, editors, *Proceedings of the International Workshop on Description Logics*, Rome, 1995.

[Buchheit *et al.* 1994] M. Buchheit, M. A. Jeusfeld, W. Nutt, and M. Staudt:. Subsumption of queries over object-oriented databases. *Information Systems*, 19(1), 1994.

[Calvanese *et al.* 1994] D. Calvanese, M. Lenzerini, and D. Nardi. A unified framework for class based representation formalisms. In J. Doyle, E. Sandewall, and P. Torasso, editors, *Proceedings of the Fourth International Conference on the Principles of Knowledge Representation and Reasoning (KR-94)*, pages 109–120, Bonn, 1994. Morgan Kaufmann, Los Altos.

[Calvanese *et al.* 1995] D. Calvanese, G. De Giacomo, and M. Lenzerini. Structured objects: Modeling and reasoning. In *Proceedings of the Fourth International Conference on Deductive and Object-Oriented Databases (DOOD-95)*, volume 1013 of *Lecture Notes in Computer Science*, pages 229–246, 1995.

[Calvanese 1996] D. Calvanese. Finite model reasoning in description logics. In *Proceedings of the Fifth International Conference on the Principles of Knowledge Representation and Reasoning (KR-96)*. Morgan Kaufmann, Los Altos, 1996.

[Calvanese & Lenzerini 1994] D. Calvanese and M. Lenzerini. Making object-oriented schemas more expressive. In *Proceedings of the Thirteenth ACM SIGACT SIGMOD Symposium on Principles of Database Systems (PODS-94)*, pages 243–254, Minneapolis, 1994. ACM Press and Addison Wesley.

[De Giacomo *et al.* 1996] G. De Giacomo, F. Donini, and F. Massacci. Exptime tableaux for $\mathcal{ALC}$. In Padgham et al. [1996].

[De Giacomo 1995] G. De Giacomo. *Decidability of Class-Based Knowledge Representation Formalisms.* PhD thesis, Università degli Studi di Roma "La Sapienza", 1995.

[De Giacomo & Lenzerini 1994a] G. De Giacomo and M. Lenzerini. Boosting the correspondence between description logics and propositional dynamic logics (extended abstract). In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, 1994.

[De Giacomo & Lenzerini 1994b] G. De Giacomo and M. Lenzerini. Concept language with number restrictions and fixpoints, and its relationship with mu-calculus. In *Proceedings of the Eleventh European Conference on Artificial Intelligence (ECAI-94)*, 1994.

[De Giacomo & Lenzerini 1995] G. De Giacomo and M. Lenzerini. What's in an aggregate: Foundations for description logics with tuples and sets. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, 1995.

[De Giacomo & Lenzerini 1996] G. De Giacomo and M. Lenzerini. Tbox and Abox reasoning in expressive description logics. In *Proceedings of the Fifth International Conference on the Principles of Knowledge Representation and Reasoning (KR-96)*, pages 316–327. Morgan Kaufmann, Los Altos, 1996.

[Donini *et al.* 1991a] F. Donini, M. Lenzerini, D. Nardi, and W. Nutt. The complexity of concept languages. In *Proceedings of the Second International Conference on the Principles of Knowledge Representation and Reasoning (KR-91)*, Boston, MA, USA, 1991.

[Donini *et al.* 1991b] F. M. Donini, M. Lenzerini, D. Nardi, and W. Nutt. Tractable concept languages. In *Proceedings of the Twelfth International*

Joint Conference on Artificial Intelligence (IJCAI-91), pages 458–463, Sydney, 1991.

[Donini *et al.* 1995] F. M. Donini, M. Lenzerini, D. Nardi, and W. Nutt. The complexity of concept languages. Technical Report RR-95-07, Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), Kaiserslautern, Germany, 1995.

[Donini *et al.* 1996] F. M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. Reasoning in description logics. In G. Brewka, editor, *Foundation of Knowledge Representation*. CSLI Publication, Cambridge University Press, 1996.

[Fischer & Ladner 1979] M. J. Fischer and R. E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Science*, 18:194–211, 1979.

[Franconi 1994] E. Franconi. A treatment of plurals and plural quantifications based on a theory of collections. *Minds and Machines*, 3(4):453–474, November 1994.

[Gen 1995] Gensym Corporation, 125 Cambridge Park Drive, Cambridge, MA. *G2 Reference Manual for G2 version 4.0.*, 1995.

[Gerstl & Pribbenow 1993] P. Gerstl and S. Pribbenow. Midwinters, end games and bodyparts. In N. Guarino and R. Poli, editors, *International Workshop on Formal Ontology-93*, pages 251–260, 1993.

[Giunchiglia & Sebastiani 1996] F. Giunchiglia and R. Sebastiani. Building decision procedures for modal logics from propositional decision procedures - the case study of modal K. In *Proceedings of the 13th Conference on Automated Deduction (CADE-96)*, Lecture Notes in Mathematics, New Brunswick, NJ, USA, 1996.

[Goñi *et al.* 1996] A. Goñi, J. Bermúdez, J. M. Blanco, and A. Illarramendi. Using reasoning of description logics for query processing in multidatabase systems. In *Working Notes of the ECAI-96 Workshop on Knowledge Representation Meets Databases (KRDB-96)*, 1996.

[Grädel *et al.* 1997] E. Grädel, M. Otto, and E. Rosen. Two-variable logic with counting is decidable. In *Proceedings of the Twelfth Annual IEEE Symposium on Logic in Computer Science (LICS-97)*, 1997. Available via `http://speedy.informatik.rwth-aachen.de/WWW/papers.html`.

[Halpern & Moses 1992] J. Y. Halpern and Y. Moses. A guide to completeness and complexity for modal logic of knowledge and belief. *Artificial Intelligence*, 54:319–379, 1992.

[Hanschke 1992] P. Hanschke. Specifying Role Interaction in Concept Languages. In *Proceedings of the Second International Conference on the Principles of Knowledge Representation and Reasoning (KR-91)*, 1992.

[Hollunder *et al.* 1990] B. Hollunder, W. Nutt, and M. Schmidt-Schauss. Subsumption algorithms for concept description languages. In *ECAI-90*, Pitman Publishing, London, 1990.

[Hollunder 1994] B. Hollunder. *Algorithmic Foundations of Terminological Knowledge Representation Systems.* PhD thesis, Universität des Saarlandes, 1994.

[Hollunder & Baader 1991] B. Hollunder and F. Baader. Qualifying number restrictions in concept languages. In *Proceedings of the Second International Conference on the Principles of Knowledge Representation and Reasoning (KR-91)*, pages 335–346, Boston, MA, USA, 1991.

[Hopcroft & Ullman 1997] J. E. Hopcroft and J. D. Ullman. *Introduction to automata theory, languages, and computation.* Addison Wesley Publ. Co., Reading, Massachussetts, 1997.

[Horrocks 1997] I. Horrocks. Optimisation techniques for expressive description logics. Technical Report UMCS-97-2-1, University of Manchester, Department of Computer Science, February 1997.

[Horrocks & Gough 1997] I. Horrocks and G. Gough. Description logics with transitive roles. In M.-C. Rousset, R. Brachmann, F. Donini, E. Franconi, I. Horrocks, and A. Levy, editors, *Proceedings of the International Workshop on Description Logics*, pages 25–28, Gif sur Yvette, France, 1997. Université Paris-Sud.

[Horrocks & Rector 1996] I. Horrocks and A. Rector. Using a description logic with concept inclusions. In Padgham et al. [1996], pages 132–135.

[Iris *et al.* 1988] M. A. Iris, B. E. Litowitz, and M. Evans. Problems of the part-whole relation. In M. W. Evans, editor, *Relational models of the lexicon*, pages 261–288. Cambridge University Press, Studies in Natural Language Processing, 1988.

[Jarke & Marquardt 1995] M. Jarke and W. Marquardt. Design and evaluation of computer-aided process modeling tools. In J. Davis, G. Stephanopoulos, and V. Venkatasubramanian, editors, *Intelligent Systems in Process Engineering, AIChE Symposium*, pages 97–109, Snowmass, CO, USA, 1995.

[Kaczmarek *et al.* 1986] T. Kaczmarek, R. Bates, and G. Robins. Recent developments in NIKL. In T. Kehler and S. Rosenschein, editors, *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, pages 978–985, CA, USA, 1986. Morgan Kaufmann, Los Altos.

[Kirk *et al.* 1995] T. Kirk, A. Y. Levy, Y. Sagiv, and D. Srivastava. The information manifold. In *Proceedings of the AAAI Spring Symposium on Information Gathering in Distributed Heterogeneous Environments*, CA, USA, 1995.

[Knuth 1968] D. Knuth. *The Art of computer programming*, volume 1. Addison Wesley Publ. Co., Reading, Massachussetts, 1968.

[Krobb 1997] C. Krobb. Entwicklung einer Spezialisierungshierarchie für Modellierungsschritte im objekt-orientierten Datenmodell VeDa. Diploma thesis, RWTH Aachen, Germany, 1997.

[Ladner 1977] R. Ladner. The computational complexity of provability in systems of modal propositional logic. *SIAM Journal of Computing*, 6(3):467–480, 1977.

[Lenzerini & Schaerf 1991] M. Lenzerini and A. Schaerf. Concept languages as query languages. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, pages 471–476, 1991.

[Leśniewski 1929] S. Leśniewski. Grundzüge eines neuen Systems der Grundlagen der Mathematik. *Fundamenta Mathematicae*, 14:1–81, 1929.

[Levy *et al.* 1996] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proceedings of the 22. International Conference on Very Large Data Bases (VLDB-96)*, Bombay, India, 1996.

[Lohmann 1998] B. Lohmann. *Ansätze zur Unterstützung des Modellierungsablaufes bei der rechnerbasierten Modellierung verfahrenstechnischer Prozesse.* PhD thesis, Lehrstuhl für Prozeßtechnik, RWTH Aachen, 1998.

[MacGregor 1991] R. MacGregor. Inside the LOOM description classifier. *SIGART Bulletin*, 2(3):88–92, 1991.

[MacGregor & Brill 1992] R. MacGregor and D. Brill. Recognition algorithms for the LOOM classifier. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pages 774–779, 1992.

[Marquardt 1992] W. Marquardt. An object-oriented representation of structured process models. In R. Gani, editor, *Proceedings of the European Symposium on Computer Aided Process Engineering*, Supplement to *Comput. Chem. Engng.* 16, pages 329–336. Elsinore, Denmark, 1992.

[Marquardt 1996] W. Marquardt. Trends in computer-aided process modeling. *Computers and Chemical Engineering*, 20(6/7):591–609, 1996.

[Molitor 1997] R. Molitor. Konsistenz von Wissensbasen in Beschreibungslogiken mit Rollenoperatoren. Diploma thesis, RWTH Aachen, Germany, 1997.

[Mortimer 1975] M. Mortimer. On languages with two variables. *Zeitschr. f. math. Logik u. Grundlagen d. Math.*, 21:135–140, 1975.

[Nebel 1988] B. Nebel. Computational complexity of terminological reasoning in BACK. *Artificial Intelligence*, 34(3):371–383, 1988.

[Nebel 1990] B. Nebel. *Reasoning and Revision in Hybrid Representation Systems*. Lecture Notes in Mathematics. Springer-Verlag, 1990.

[Pacholski *et al.* 1997] L. Pacholski, W. Szwast, and L. Tendera. Complexity of two-variable logic with counting. In *Proceedings of the Twelfth Annual IEEE Symposium on Logic in Computer Science (LICS-97)*, 1997.

[Padgham *et al.* 1996] L. Padgham, E. Franconi, M. Gehrke, D. L. McGuinness, and P. F. Patel-Schneider, editors. *Proceedings of the International Workshop on Description Logics*, Cambridge (Boston), MA, U.S.A., 1996. AAAI Press/The MIT Press.

[Padgham & Lambrix 1994] L. Padgham and P. Lambrix. A framework for part-of hierarchies in terminological logics. In J. Doyle, E. Sandewall, and P. Torasso, editors, *Proceedings of the Fourth International Conference on the Principles of Knowledge Representation and Reasoning (KR-94)*, pages 485–496, 1994.

[Patel-Schneider *et al.* 1991] P. Patel-Schneider, D. McGuinness, R. Brachman, L. Resnick, and A. Borgida. The CLASSIC knowledge representation system: Guiding principles and implementation rationale. *SIGART Bulletin*, 2(3):108–113, 1991.

[Patel-Schneider 1989] P. F. Patel-Schneider. Undecidability of subsumption in NIKL. *Artificial Intelligence Journal*, 39:263–272, 1989.

[Peltason 1991] C. Peltason. The BACK System - an Overview. *SIGART Bulletin*, 2(3):114–119, 1991.

[Pohl 1995] K. Pohl. *A process centered requirements engineering environment.* PhD thesis, Lehrstuhl für Informatik V, RWTH Aachen, 1995.

[Pratt 1979] V. R. Pratt. Models of program logics. In *Proceedings of the 20th Annual Symposium on Foundations of Computer Science*, San Juan, Puerto Rico, 1979.

[Pribbenow 1995] S. Pribbenow. Modeling physical objects: Reasoning about (different kinds of) parts. In *Time, Space, and Movement Workshop 95*, Bonas, France, 1995.

[Rychtyckyj 1996] N. Rychtyckyj. DLMS: An evaluation of KL-ONE in the automobile industry. In *Proceedings of the Fifth International Conference on the Principles of Knowledge Representation and Reasoning (KR-96)*, Boston, MA, USA, 1996. Morgan Kaufmann, Los Altos.

[Sattler 1996] U. Sattler. A concept language extended with different kinds of transitive roles. In G. Görz and S. Hölldobler, editors, *20. Deutsche Jahrestagung für Künstliche Intelligenz*, volume 1137 of *Lecture Notes in Mathematics*. Springer-Verlag, 1996.

[Savitch 1970] W. J. Savitch. Relationsship between nondeterministic and deterministic tape complexities. *Journal of Computer and System Science*, 4:177–192, 1970.

[Schild 1991] K. Schild. A correspondence theory for terminological logics: Preliminary report. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 466–471, Sydney, 1991.

[Schild 1994] K. Schild. Terminological cycles and the propositional μ-calculus. In J. Doyle, E. Sandewall, and P. Torasso, editors, *Proceedings of the Fourth International Conference on the Principles of Knowledge Representation and Reasoning (KR-94)*, pages 509–520, Bonn, 1994. Morgan Kaufmann, Los Altos.

[Schmidt-Schauss 1989] M. Schmidt-Schauss. Subsumption in KL-ONE is undecidable. In *Proceedings of the First International Conference on the Principles of Knowledge Representation and Reasoning (KR-89)*, pages 421–431, Boston (USA), 1989.

[Schmidt-Schauß & Smolka 1991] M. Schmidt-Schauß and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.

[Simons 1987] P. M. Simons. *Parts. A study in Ontology.* Oxford: Clarendon, 1987.

[Souza 1998] D. Souza. VeDa - part 2: Structural Modeling Objects. Technical report, Lehrstuhl für Prozeßtechnik RWTH Aachen, 1998. To appear.

[Sowa 1987] J. F. Sowa. Semantic networks. In S. C. Shapiro, editor, *Encyclopedia of Artificial Intelligence 2.* John Wiley & Sons, New York, 1987.

[Thomas 1992] W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of theoretical computer science*, volume B. Elsevier Science Publishers (North-Holland), Amsterdam, 1992.

[Wang 1963] H. Wang. Dominoes and the AEA case of the Decision Problem. *Bell Syst. Tech. J.*, 40:1–41, 1963.

[Winston *et al.* 1987] M. Winston, R. Chaffin, and D. Herrmann. A taxonomy of part whole relations. *Cognitive Science*, 11:417–444, 1987.

Index

List of Figures