Techniques for Optimized Reasoning in Description Logic Knowledge Bases

Claudia Schon

Juni 2016

Vom Promotionsausschuss des Fachbereichs 4: Informatik der
Universität Koblenz-Landau zur Verleihung des akademischen
Grades **Doktor der Naturwissenschaften (Dr. rer. nat.)**
genehmigte Dissertation

Vorsitzender des Promotionsausschusses:	Prof. Dr. Ralf Lämmel
Vorsitzender der Promotionskommission:	Prof. Dr. Dietrich Paulus
1. Berichterstatter:	Prof. Dr. Ulrich Furbach
2. Berichterstatter:	Prof. Dr. Steffen Staab

Datum der wissenschaftlichen Aussprache: 15.06.2016

Veröffentlicht als Dissertation an der Universität Koblenz-Landau.

Abstract

One of the main goals of the artificial intelligence community is to create machines able to reason with dynamically changing knowledge. To achieve this goal, a multitude of different problems have to be solved, of which many have been addressed in the various sub-disciplines of artificial intelligence, like automated reasoning and machine learning. The thesis at hand focuses on the automated reasoning aspects of these problems and address two of the problems which have to be overcome to reach the afore-mentioned goal, namely 1. the fact that reasoning in logical knowledge bases is intractable and 2. the fact that applying changes to formalized knowledge can easily introduce inconsistencies, which leads to unwanted results in most scenarios.

To ease the intractability of logical reasoning, I suggest to adapt a technique called knowledge compilation, known from propositional logic, to description logic knowledge bases. The basic idea of this technique is to compile the given knowledge base into a normal form which allows to answer queries efficiently. This compilation step is very expensive but has to be performed only once and as soon as the result of this step is used to answer many queries, the expensive compilation step gets worthwhile. In the thesis at hand, I develop a normal form, called linkless normal form, suitable for knowledge compilation for description logic knowledge bases. From a computational point of view, the linkless normal form has very nice properties which are introduced in this thesis.

For the second problem, I focus on changes occurring on the instance level of description logic knowledge bases. I introduce three change operators interesting for these knowledge bases, namely deletion and insertion of assertions as well as repair of inconsistent instance bases. These change operators are defined such that in all three cases, the resulting knowledge base is ensured to be consistent and changes performed to the knowledge base are minimal. This allows us to preserve as much of the original knowledge base as possible. Furthermore, I show how these changes can be applied by using a transformation of the knowledge base.

For both issues I suggest to adapt techniques successfully used in other logics to get promising methods for description logic knowledge bases.

Zusammenfassung

Ein Hauptziel des Forschungsbereiches der Künstlichen Intelligenz ist das Entwickeln einer Maschine, die in der Lage ist logische Schlussfolgerungen aus großen Wissensbasen zu ziehen, die sich dynamisch verändern. Um dieses Ziel erreichen zu können, muss eine Vielzahl von Probleme gelöst werden, die in verschiedenen Teildisziplinen der Künstlichen Intelligenz, wie zum Beispiel dem automatischen Schließen und dem maschinelles Lernen, liegen. Diese Dissertation konzentriert sich auf die folgenden zwei Hindernisse aus dem Bereich des automatischen Schließens: 1. Die Tatsache, dass Schließen in logischen Wissensbasen sehr aufwendig sein kann und 2. die Tatsache, dass Änderungen an formalisiertem Wissen häufig Inkonsistenzen hervorrufen, die beim Schließen zu unerwünschten Ergebnissen führen können.

Um das Schließen in Wissensbasen, die in einer Beschreibungslogik gegeben sind, effizienter zu machen, schlage ich vor, die Technik der Knowledge Compilation zu verwenden, die bereits erfolgreich in der Aussagenlogik verwendet wird und die ich entsprechend weiterentwickelt habe. Dabei wird die gegebene Wissensbasis einmalig in eine Normalform umgewandelt, die es erlaubt Anfragen schneller als in der ursprünglichen Form zu verarbeiten. Zwar ist dieser Umwandlungsschritt zeitaufwendig, doch sobald mehrere Anfragen an die umgewandelte Wissensbasis gestellt werden, zahlt sich dieser Mehraufwand durch die verkürzten Antwortzeiten aus. Die Normalform, die ich im Rahmen meiner Forschungsarbeit entwickelt habe, ist die sogenannte Linkless Normalform für Beschreibungslogik, deren zahlreiche interessanten Eigenschaften in dieser Dissertation vorgestellt werden.

Bei der Entwicklung einer möglichen Lösung für das zweiten Problems konzentriere ich mich auf Veränderungen auf der Instanzebene von Wissensbasen, die in einer Beschreibungslogik vorliegen. Dafür habe ich drei Operatoren entwickelt, die es erlauben die Instanzebene dieser Wissensbasen zu verändern: das Löschen und Hinzufügen von Informationen sowie das Reparieren von inkonsistenten Instanzebenen. Diese drei Operatoren sind so definiert, dass sichergestellt wird, dass die Wissensbasis nach Anwendung eines solchen Operators stets konsistent ist. Um möglichst wenig formalisiertes Wissen aus der Wissensbasis zu verlieren, ist bei allen drei Operatoren sichergestellt, dass die an der Wissensbasis durchgeführten Änderungen minimal sind. Die eigentliche Anwendung dieser drei Operatoren lässt sich dabei über eine Transformation der Wissensbasis erreichen, die ich ebenfalls auf Grundlage einer Transformation aus der Aussagenlogik entwickelt habe.

Bei beiden Problemen konzentriere ich mich auf in einer Beschreibungslogik gegebene Wissensbasen und verwende Techniken, die bereits in anderen Logiken erfolgreich eingesetzt werden, um Lösungen zu bieten.

Acknowledgements

There are many people who helped me along the way during the creation of this thesis and to whom I would like to express my gratitude.

First of all I gratefully acknowledge all the advice, useful critique and encouragement my supervisor, Ulrich Furbach, provided over the past few years. His door was always open whenever I had questions or needed someone to discuss my ideas. With his enthusiasm for research he inspired me again and again. It was and still is a great pleasure to work with him.

I would like to thank Steffen Staab for kindly accepting to review this thesis and for providing the opportunity to present my work to his group.

To my friend and colleague Markus Bender, thank you for taking the task of proofreading upon yourself and for providing constructive feedback. I really appreciate all your help. To Matthias Horbach, thank you for providing constructive critique to parts of this thesis as well as all the coffee breaks, insightful discussions and visits to the complexity zoo together with Markus Bender and Christian Schwarz. To Sonja Polster and Heiko Günther for contributing by implementing parts of my approaches. Furthermore, to all former members of the artificial intelligence research group, thank you for all the insightful discussions and the friendly atmosphere in the group.

I would like to thank the Universität Koblenz-Landau as well as the Deutsche Forschungsgemeinschaft (DFG) for supporting my research.

To my parents and my mother in law, thank you for your continuous support by babysitting my children in times of need. And of course to Norman, thank you for not only backing me up and being there for me during the stressful moments, but also for sharing all the good moments. Last but not least, to my children, thank you for continuously providing joyful distractions.

Contents

List of Figures

List of Tables

1 Introduction

It is a ongoing dream of the artificial intelligence community to create machines able to reason with dynamically changing knowledge. Even though many steps towards this vision were made, still many obstacles have to be overcome. In this thesis we tackle two of these obstacles: 1. the fact that logical reasoning is intractable and 2. the fact that applying changes to formalized knowledge can easily introduce inconsistencies making it difficult to reason properly. We suggest to ease the intractability of logical reasoning by using a technique called knowledge compilation. This technique consists of the compilation of the knowledge base into a special form which can be used to perform certain reasoning tasks efficiently. To overcome the second obstacle, we develop techniques for applying changes to the instance level of description logic knowledge bases. These techniques ensure that the knowledge base resulting from a change is consistent thus avoiding the problems of inconsistencies.

1.1 Used Techniques

Knowledge representation is an area of artificial intelligence dealing with the formal representation of information. One possibility to formalize knowledge is the use of *logical knowledge bases*, in which the represented information can be accessed by deductive reasoning methods. This makes it possible not only to access knowledge that is explicitly stated but also implicitly stated knowledge.

There are numerous logical formalisms suitable for knowledge representation purposes, such as propositional logic, first-order logic, modal logics and description logics. In the context of knowledge representation, it is not only of interest to adequately formalize the information: another important aspect is the development of efficient reasoning procedures working on the representations. Given for example a knowledge base KB and a formula F, it is of interest if F is a *logical consequence* of KB. As soon as the knowledge base is subject to changes, an additional challenge arises: Changes can easily introduce inconsistencies into a knowledge base causing every formula F to be a consequence of the knowledge base. For each of the above-mentioned logics, a large number of formalisms and reasoning procedures have been developed throughout the years. The question suggests itself, whether methods developed in one of these logics can be adapted to work in another logic. This idea allows to partially reuse approaches such that a procedure does not have to be developed from scratch. The idea of reusing methods developed for one logic for another logic is not new. Consider for example the Davis–Putnam–Logemann–Loveland (DPLL) algorithm (Davis and Putnam, 1960) which is an algorithm for deciding the satisfiability of propositional logic formulae. This algorithm was extended to a gen-

eral procedure DPLL(X) (Ganzinger, Hagen, Nieuwenhuis, Oliveras, and Tinelli, 2004) allowing DPLL to be extended with a solver for a theory given in first-order logic.

This reuse of approaches is not limited to transferring methods from one logic to another one but it is also beneficial to reuse research within one logic. An impressive example is the use of unification, which was introduced for first-order resolution and later applied in first-order logic tableaux. In this thesis, two formalisms from different logics are extended to description logic.

- A normal form known from the area of knowledge compilation in propositional logic is adapted to description logics, allowing to perform certain reasoning tasks efficiently.
- An approach for view deletion in deductive databases is adapted to change methods for the instance level of description logic knowledge bases.

Naturally, the thesis at hand used different logic. This is why, in the next section we give a short overview on the different logics playing an important role in this thesis.

1.2 Logics

Besides propositional logic and first-order logic, description logics are important in the scope of this thesis. Description logics are a family of knowledge representation languages used to model knowledge. A description logic knowledge base consists of a terminological part, called TBox, formalizing knowledge about the world and an assertional part, called ABox, containing information on individuals, concepts the individuals belong to and relations between individuals. For some description logics, knowledge bases also contain an RBox, formalizing knowledge about relations. There is a variety of description logics offering different degrees of expressivity in which most of the description logics are located between propositional and first-order logic considering their expressivity while still being decidable. In this thesis, the description logics $\mathcal{ALC}$, $\mathcal{ALE}$ and $\mathcal{SHI}$ come to use.

1.3 Knowledge Compilation

In the first part of this thesis, the idea of a normal form for propositional logic called *linkless normal form* is applied to concepts and TBoxes given in the description logic $\mathcal{ALC}$. In propositional logic, this normal form is used in the area of knowledge compilation (Murray and Rosenthal, 1985). Knowledge compilation is a technique used to deal with the computational intractability of deduction in logical formalisms. It is a well investigated technique for knowledge bases given in propositional logic (Darwiche and Marquis, 2002; Cadoli and Donini, 1997). Consider the following task: For a propositional logic knowledge base KB and a formula F, we want to check if F is a logical consequence of KB. It is well-known that this test has an exponential time complexity, namely exponential to the number of distinct variables in KB and F. The basic idea of knowledge compilation is to divide the reasoning process into two phases: an off-line

phase and an online phase. During the off-line phase, the knowledge base is compiled into a certain normal form. This compilation usually is very demanding from a computational point of view but has to be performed only once. During the online phase, the precompiled version of the knowledge base is used to answer queries. In our example to check if F is a logical consequence. For answering just one query, the effort of performing this expensive precompilation step by no means justifies the benefit. Let us assume that the task is not to only check if one single formula F is entailed by KB but for a large set $\{F_1, \ldots, F_n\}$ of formulae. In such a setting, the expenses of the precompilation step can be spread over the different queries making this step worthwhile. Linkless normal form is one normal form which lends itself for the compilation step of propositional logic formulae. Propositional logic formulae given in linkless normal form possess some nice properties such that they allow constant time satisfiability test and they can be efficiently projected on a set of atoms. The thought of having such a normal form for description logic concepts and TBoxes sounds tempting. This is why this thesis adapts the idea of the linkless normal form to the description logic $\mathcal{ALC}$ such that the resulting normal form also offers interesting properties.

1.4 Evolution of the Instance Level of Knowledge Bases

In the second part of this thesis, a successful approach for view deletion in deductive databases (Aravindan and Baumgartner, 2000) is applied for implementing changes into description logic ABoxes. In practical use, knowledge bases are often not fixed but subject to changes. In the scenario examined in this thesis, the TBox of the knowledge base is considered to be fixed in contrast to the ABox which is confronted with changes. The changes to the ABox that are considered in this thesis are insertion of an assertion, deletion of an assertion and repair of an ABox which is inconsistent with respect to its TBox. Performing changes to an ABox can easily render an ABox inconsistent with respect to its TBox. Consider for example a TBox stating that no one can be both female and male

$$\mathcal{T} = \{Female \sqsubseteq \neg Male\}$$

together with an ABox

$$\mathcal{A} = \{Female(sasha)\}.$$

Adding the assertion $Male(sasha)$ to $\mathcal{A}$ renders it inconsistent with respect to $\mathcal{T}$. From a reasoning point of view, an inconsistent knowledge base is problematic since it is possible to deduce everything from such a knowledge base with common methods. Therefore, it is not only desirable to repair an ABox which is inconsistent with respect to its TBox but it is important that the methods for changing an ABox developed in this thesis always result in an ABox which is consistent with respect to its TBox. On the other hand it is desirable that a change affects as little of the knowledge base as possible. All change operations developed in this thesis are defined such that only a minimal set of ABox assertions is actually changed. For the computation of the change operations, a transformation similar to the technique used by Aravindan and Baumgartner (2000)

for deductive databases is used. There the idea is to transform the fixed part of the deductive database before applying the methods for changing the knowledge base. This transformation is guided by the set of ground facts occurring in the knowledge base. One can label this transformation as *semantically guided* since the set of ground facts can be considered to be a (partial) model of the deductive database. This semantical guidance reflects the assumption that the ground facts of the deductive database are regarded to be true and allows to compute only the deviation from the original set of ground facts. When considering deductive databases with a large number of ground facts, it is reasonable to assume that changes only affect very few ground facts. This assumption makes the semantical guidance of the transformation especially advantageous. When considering changes to description logic ABoxes, we are facing a similar setting: It is likely that the ABox constitutes at least a partial model of the TBox and furthermore it is reasonable to assume that changing a consistent ABox as well as repairing an inconsistent ABox only affects very few assertions. These similarities suggest that it is beneficial to adapt the ideas introduced by Aravindan and Baumgartner (2000) to description logic ABoxes as well.

Next, we present a brief overview on the different chapters of the thesis at hand.

1.5 Overview

Chapter 2 provides the necessary preliminaries for the topics addressed in this thesis. We introduce syntax and semantics of propositional logic (Section 2.1), first-order logic (Section 2.2), multi-modal logic K_n (Section 2.3) and the description logics $\mathcal{ALE}$, $\mathcal{ALC}$ and $\mathcal{SHI}$ (Section 2.4). All these logics are used when we explore the application of established techniques to description logics.

Chapter 3 addresses the problem of knowledge compilation. As described afore, the basic idea of knowledge compilation is to compile a knowledge base during an off-line phase into a special normal form and then use the result of this precompilation to answer queries during the online phase.

Section 3.1 concentrates on well-known techniques for knowledge compilation in propositional logic and introduces ideas how to compare different normal forms together with a collection of normal forms suitable for the precompilation step of knowledge compilation. The normal forms introduced are the decomposable negation normal form, the linkless normal form, prime implicant normal form and prime implicat normal form. This presentation of normal forms does not strive for completeness. It contains only those normal forms interesting in the scope of this thesis.

Section 3.2 applies established techniques from the area of knowledge compilation in propositional logic to knowledge bases given in description logics. One main contribution of this thesis is the development of several normal forms for concepts given in the description logic $\mathcal{ALC}$: the $\forall$- and $\exists$-normal form are introduced as a basis of the linkless normal form for $\mathcal{ALC}$ concepts which is developed in Section 3.2.2. Properties of the linkless normal form are presented together with proofs. Furthermore, it is shown

how to use this normal form for the precompilation of $\mathcal{ALC}$ TBoxes which makes the linkless normal form an interesting candidate for the precompilation step of knowledge compilation for knowledge bases given in $\mathcal{ALC}$.

A prototypical implementation which is able to transform $\mathcal{ALC}$ TBoxes into linkless normal form was developed and Section 3.2.3.1 presents of some experimental results.

Section 3.2.4 arranges the developed linkless normal form into the line of related work and Section 3.3 introduces topics interesting for future research in the scope of the linkless normal form.

Chapter 4 addresses the task of changing the instance level of a knowledge base. For this, in Section 4.1 the problem of view deletion in deductive databases is introduced. Aravindan and Baumgartner (2000) developed an approach to solve this problem using theorem proving techniques. To conduct a change to the deductive database, their method only computes the deviation from the original instance base of the knowledge base. As soon as the knowledge base contains a very large instance base, this approach becomes beneficial, since a growing size of the instance base does not have too much influence on the efficiency of such an approach. Since in description logic there are many knowledge bases with rather large ABoxes, it sounds tempting to have a method for performing changes to an ABox by only computing the deviation from the original ABox.

A main contribution of this thesis is presented in Section 4.2, where the approach for view deletion in deductive databases is adapted to the description logic $\mathcal{SHI}$: Section 4.2.1 introduces three different change operators for $\mathcal{SHI}$ knowledge bases, namely *deletion* and *insertion* of assertions and *repair* of an ABox which is inconsistent w.r.t. its TBox.

Section 4.2.2 presents the so-called $\mathcal{K}^*$-transformation which is used to transform the TBox of the knowledge base under consideration into a special form. This transformation is guided by the ABox on the basis of the assumption that the ABox is already very close to a model of the TBox.

Section 4.2.3 shows that result of the $\mathcal{K}^*$-transformation can be used to efficiently compute the result of the afore-introduced changing methods and provides proofs that this actually computes the desired outcome.

The $\mathcal{K}^*$-transformation is implemented in a system which can be used to perform the introduced changing methods on $\mathcal{SHI}$ knowledge bases. Section 4.2.4 presents experimental results for all change operations and shows: The efficiency of the approach is not much influenced by the size of the ABox but rather by the number of assertions affected by the change operation.

Section 4.3 arranges the developed approach for the computation of changes to the instance level of $\mathcal{SHI}$ knowledge bases into the line of related work. The chapter ends with Section 4.4 pointing out interesting avenues for future research.

Chapter 5 summarizes the methods introduced in Chapter 3 and Chapter 4 and gives a réumé of the main contributions of this thesis.

1.5.1 Contributions

My overall contributions can be summarized as follows:

1. For knowledge bases given in propositional logic, knowledge compilation is a well-known technique and there are numerous normal forms which are commonly used for the precompilation step. Even though knowledge compilation is a tempting approach for description logic knowledge bases, to the best of my knowledge, this area has hardly been investigated. I ease this situation by developing several normal forms for $\mathcal{ALC}$ concepts like the so called $\forall$-*normal form* and the $\exists$-*normal form* which serve as a basis for the development of the so called *linkless normal form*. The linkless normal form is introduced both for $\mathcal{ALC}$ concepts and TBoxes and is suitable for knowledge compilation purposes. Properties of the linkless normal form are presented along with proofs.

2. There are approaches performing changes to the instance level of knowledge bases given in the description logic DL-Lite. However these techniques cannot be applied to more expressive description logics like the description logic $\mathcal{SHI}$ and I am not aware of any other approaches to tackle this task in $\mathcal{SHI}$. In this thesis, I develop a method for performing changes to the instance level of knowlege bases given in the description logic $\mathcal{SHI}$. For this a transformation of the knowledge base, the so called $\mathcal{K}^*$-*transformation* is introduced which is guided by the ABox of the knowledge base. The result of this transformation can be used to efficiently perform changes to the ABox of the knowledge base:

   - insertion of assertions,
   - deletion of assertions and
   - repair.

   For each of the three changes it is ensured that only a minimal set of ABox assertions is affected by the change. Furthermore, this method only computed the deviation form the original ABox making this method particularly advantageous for change scenarios with large ABoxes where only few ABox assertions are affected by the changes.

The main contributions of this thesis are already published:

- Many results from Chapter 3 were published in (Schon, 2011, 2010; Furbach, Günther, and Obermaier, 2009; Furbach and Obermaier, 2007a) and very preliminary work in (Furbach and Obermaier, 2007b).

- The results from Chapter 4 were published in (Furbach and Schon, 2013a) and in at full length in (Furbach and Schon, 2013b).

2 Preliminaries

This chapter introduces the different logics necessary to understand this thesis. Firstly, in Section 2.1 propositional logic and in Section 2.2 first-order logic are introduced. Next Section 2.3 presents modal logic and Section 2.4 description logic.

The reason to present such a large selection of different logics lies in the topic of this thesis. In the main parts of the document at hand, ideas from propositional logic are extended for description logics. Hence both logics are presented in sequel. In addition to that, the methods for the evolution of the instance level of description logic knowledge bases presented in Chapter 4 use so-called DL-clauses. Their syntax is similar to first-order logic clauses. This is why besides DL-clauses, first-order logic is presented as well. Description logics are closely related to modal logic. Therefore many results of this thesis could be easily transferred to modal logic. This is why modal logic is introduced as well.

2.1 Propositional Logic

In Section 2.1.1 the syntax of propositional logic formulae is introduced, followed by Section 2.1.2 presenting semantics of these formulae. Furthermore, some equivalences and normal forms are presented in Section 2.1.3.

2.1.1 Syntax

Definition 2.1.1 (Syntax of Propositional Logic Formulae)**.** *Let V be a countable set of propositional logic variables. The set of propositional logic formulae over V, denoted by F^V_{prop}, is the smallest set satisfying the following conditions:*

- $V \subseteq F^V_{prop}$.
- $\bot \in F^V_{prop}$.
- $\top \in F^V_{prop}$.
- *If* $G \in F^V_{prop}$, *then* $\neg G \in F^V_{prop}$.
- *If* $G, H \in F^V_{prop}$, *then* $(G \wedge H) \in F^V_{prop}$.
- *If* $G, H \in F^V_{prop}$, *then* $(G \vee H) \in F^V_{prop}$.

A set of propositional logic formulae is called propositional logic knowledge base.

If the set of propositional logic variables is not of interest, V is omitted and we speak of F_{prop} instead. When convenient, notation is slightly abused by treating $\wedge$ and $\vee$ as multiarity connectives ranging from zero connected formulae to any arbitrary finite number of connected formulae. Note that the empty disjunction corresponds to $\bot$ and the empty conjunction to $\top$. Furthermore, $a \rightarrow b$ is used as a shorthand for $\neg a \vee b$.

Definition 2.1.2 (Set of Subformulae)**.** *Let F, G and H be propositional logic formulae in F^V_{prop}. $\mathsf{sub} : F^V_{prop} \rightarrow 2^{F^V_{prop}}$ is a function mapping a formula F to the smallest set of formulae satisfying the following conditions:*

- $F \in \mathsf{sub}(F)$.
- *If* $\neg G \in \mathsf{sub}(F)$ *then* $G \in \mathsf{sub}(F)$.
- *If* $G \wedge H \in \mathsf{sub}(F)$ *then* $G \in \mathsf{sub}(F)$ *and* $H \in \mathsf{sub}(F)$.
- *If* $G \vee H \in \mathsf{sub}(F)$ *then* $G \in \mathsf{sub}(F)$ *and* $H \in \mathsf{sub}(F)$.

For a formula F, $\mathsf{sub}(F)$ is called the set of subformulae of F.

A formula of the form $F = v$ for a propositional logic variable v is called *atom*. Furthermore, a formula of the form $G \vee H$ is called *disjunction* and G and H are called *disjuncts*. A formula of the form $G \wedge H$ is called *conjunction* and G and H are called *conjuncts*.

Sometimes, it is convenient to neglect the order of conjuncts in n-ary conjunctions and n-ary disjunctions of in a formula when determining the set of its subformulae. We denote these subformulae as *subformulae modulo commutativity*. When considering for example the formula $F = a \wedge b \wedge c$, the set of subformulae modulo commutativity is $\{a, b, c, a \wedge b, a \wedge c, b \wedge a, b \wedge c, c \wedge a, c \wedge b, a \wedge b \wedge c\}$.

Furthermore, the function $\mathsf{var} : F^V_{prop} \rightarrow 2^V$ is a function mapping a formula F over the set of propositional logic variables V to the set of atoms occurring as a subformula of F.

2.1.2 Semantics

Next semantics of propositional logic formulae is introduced together with the notion of a model of a formula.

Definition 2.1.3 (Semantics of Propositional Logic Formulae)**.** *Let V be a set of propositional logic variables and $\{true, false\}$ a set of truth values. Then $\mathcal{I}_V : V \rightarrow \{true, false\}$ is called an interpretation. The extension $\mathcal{I}_{F^V_{prop}} : F^V_{prop} \rightarrow \{true, false\}$ of $\mathcal{I}_V$ is called an interpretation of propositional logic formulae from F^V_{prop} and is defined by:*

- $\mathcal{I}_{F^V_{prop}}(\bot) = false$.
- $\mathcal{I}_{F^V_{prop}}(\top) = true$.
- $\mathcal{I}_{F^V_{prop}}(a) = \mathcal{I}_V(a)$, *if* $a \in V$.

- $\mathcal{I}_{F^V_{prop}}(\neg F) = \begin{cases} true & if\ \mathcal{I}_{F^V_{prop}}(F) = false \\ false & otherwise. \end{cases}$

- $\mathcal{I}_{F^V_{prop}}(F \wedge G) = \begin{cases} true & if\ \mathcal{I}_{F^V_{prop}}(F) = true\ and\ \mathcal{I}_{F^V_{prop}}(G) = true \\ false & otherwise. \end{cases}$

- $\mathcal{I}_{F^V_{prop}}(F \vee G) = \begin{cases} true & if\ \mathcal{I}_{F^V_{prop}}(F) = true\ or\ \mathcal{I}_{F^V_{prop}}(G) = true \\ false & otherwise. \end{cases}$

Note that $\mathcal{I}_V$ is contained in $I_{F^V_{prop}}$. We slightly abuse notation and use $\mathcal{I}$ instead of $\mathcal{I}_{F^V_{prop}}$ as an interpretation for both propositional logic variables and formulae. When convenient, an interpretation $\mathcal{I}$ is regarded as the set of propositional logic variables assigned to *true* by $\mathcal{I}$.

Definition 2.1.4 (Model)**.** *Let F be a propositional logic formula and $\mathcal{I}$ an interpretation. $\mathcal{I}$ is a* model *for F, denoted by $\mathcal{I} \models F$ iff $\mathcal{I}(F) = true$. Let further N be a set of propositional logic formulae. $\mathcal{I}$ is a model for N iff $\mathcal{I} \models G$ for every $G \in N$.*

Definition 2.1.5 (Satisfiable, Unsatisfiable, Valid)**.** *A propositional logic formula F is called* satisfiable *iff there is a model for F. Otherwise F is called* unsatisfiable*. F is called* valid *or tautologous if every interpretation is a model for F. A set of propositional logic formulae N is called satisfiable iff all formulae in N are satisfiable. Otherwise N is unsatisfiable.*

Note that satisfiability of a formula can be tested by checking all possible interpretations. For details on satisfiability checking of propositional logic formulae we refer the reader to Fitting (1996).

In general, there can be more than one model for a propositional logic formula. In this thesis, it is sometimes necessary to compare different models for a formula. This is why we introduce the notion of minimal models.

Definition 2.1.6 (Minimal Model)**.** *Let F be a propositional logic formula. An interpretation $\mathcal{I}$ is a minimal model for F iff $\mathcal{I}$ is a model for F and there is no model $\mathcal{I}'$ for F with $\mathcal{I}' \subset \mathcal{I}$.*

Using this notion, it is possible to compare models. A model $\mathcal{I}_1$ is called smaller than a model $\mathcal{I}_2$, if $\mathcal{I}_1 \subset \mathcal{I}_2$. Note that it is possible for two models I_1 and I_2 that neither I_1 is smaller than I_2 nor I_2 is smaller than I_1. Take $F = a \vee b$ and $\mathcal{I}_1 = \{a\}$ and $\mathcal{I}_2 = \{b\}$ as an example.

Example 2.1.7. *Given the propositional logic formula*

$$F = \neg(a \vee b \vee (e \wedge \neg f))$$

and the interpretation $\mathcal{I}$ with:

$$\mathcal{I}(a) = \mathit{false}$$
$$\mathcal{I}(b) = \mathit{false}$$
$$\mathcal{I}(e) = \mathit{true}$$
$$\mathcal{I}(f) = \mathit{true}$$

Then $\mathcal{I}(F) = \mathit{true}$, hence $\mathcal{I}$ is a model for F. As mentioned afore, $\mathcal{I}$ can be represented as the set of propositional logic variables assigned to true by $\mathcal{I}$, leading to $\mathcal{I} = \{e, f\}$.

Given a second interpretation $\mathcal{I}'$ with:

$$\mathcal{I}' = \{f\}$$

which is a model for F. Comparing $\mathcal{I}$ and $\mathcal{I}'$ shows that $\mathcal{I}' \subset \mathcal{I}$ and therefore $\mathcal{I}$ is not a minimal model for F.

2.1.3 Equivalences and Normal Forms

In the following, several normal forms for propositional logic formulae will be investigated. To translate a propositional logic formula into a normal form, it is necessary to perform transformations on given formulae. These transformations should work without changing the semantics of the formula, i.e. equivalence is supposed to be preserved during the transformation.

Definition 2.1.8 (Equivalent Formulae)**.** *Let F and G be propositional logic formulae. F and G called* semantically equivalent*, denoted by $F \equiv G$ iff for every interpretation $\mathcal{I}$: $\mathcal{I}(F) = \mathcal{I}(G)$.*

When regarding transformations on formulae, we are only interested in transformations that preserve equivalence of a formula. The basic idea of these transformations is the following: Given a formula F containing a subformula G. If this subformula G is equivalent to a formula G', then it is possible to substitute G' for G in F. The resulting formula is equivalent to the original formula F.

Theorem 2.1.9. *Let F, G' be propositional logic formulae and G be a subformula of F with $G \equiv G'$. Let further F' be the formula obtained from F by substituting G' for an occurrence of G. Then $F \equiv F'$.*

Next some equivalences are presented, which can be used to transform propositional logic formulae.

Theorem 2.1.10. *Let F, G and H be propositional logic formulae. The following equivalences hold:*

$$\neg\neg F \equiv F \text{ (Double negation)}$$

$$F \wedge F \equiv F$$
$$F \vee F \equiv F \text{ (Idempotence)}$$

$$F \wedge G \equiv G \wedge F$$
$$F \vee G \equiv G \vee F \text{ (Commutativity)}$$

$$F \wedge (G \wedge H) \equiv (F \wedge G) \wedge H$$
$$F \vee (G \vee H) \equiv (F \vee G) \vee H \text{ (Associativity)}$$

$$(F \wedge G) \vee H \equiv (F \vee H) \wedge (G \vee H)$$
$$(F \vee G) \wedge H \equiv (F \wedge H) \vee (G \wedge H) \text{ (Distributivity)}$$

$$F \wedge (F \vee G) \equiv F$$
$$F \vee (F \wedge G) \equiv F \text{ (Absorption)}$$

$$\neg(F \wedge G) \equiv \neg F \vee \neg G$$
$$\neg(F \vee G) \equiv \neg F \wedge \neg G \text{ (De Morgan's laws)}$$

$$F \vee G \equiv F \text{ if } F \text{ is a tautology}$$
$$F \wedge G \equiv G \text{ if } F \text{ is a tautology}$$
$$F \vee G \equiv G \text{ if } F \text{ is unsatisfiable}$$
$$F \wedge G \equiv F \text{ if } F \text{ is unsatisfiable}$$

Proof of Theorem 2.1.10 can be done using truth tables.

The equivalences presented in Theorem 2.1.10 can be used to transform any propositional logic formula into three basic normal forms defined as follows:

Definition 2.1.11 (Negation Normal Form)**.** *A propositional logic formula F is in* negation normal form (NNF) *iff the negation symbol only occurs in front of propositional logic variables.*

It is possible to transform every propositional logic formula into NNF by removing double negations and moving the negations as far as possible to the inside of the formula using De Morgan's laws.

A *literal* denotes an atom or a negated atom. Two literals like b and $\neg b$ using the same atom but having different polarity are called *complementary literals*. For a literal L, we use $\overline{L}$ to denote the respective complementary literal.

Definition 2.1.12 (Clausal Normal Form)**.** *A* clause *is a disjunction of literals. A propositional logic formula is in* conjunctive normal form *or* clausal normal form *(CNF) iff it is a conjunction of clauses. For a formula F in CNF with*

$$F = (L_{1,1} \vee \ldots \vee L_{1,n_1}) \wedge \ldots \wedge (L_{k,1} \vee \ldots \vee L_{k,n_k})$$

where $L_{i,j}$ are literals. Often set notation is used, where clauses are represented as sets of literals and the CNF as a set of clauses:

$$F = \{\{L_{1,1}, \ldots, L_{1,n_1}\}, \ldots, \{L_{k,1}, \ldots, L_{k,n_k}\}\}$$

Note that if a disjunction of literals has zero arity, this corresponds to the empty clause written as $\{\}$.

Definition 2.1.13 (Horn Formula)**.** *A clause is called* Horn clause *if it contains at most one positive literal. A formula given in CNF is a* Horn formula *if all its clauses are Horn clauses.*

Horn formula have very nice computational properties. As shown by Dowling and Gallier (1984), satisfiability of a Horn formula can be decided in time linear to the total number of occurrences of literals in the formula.

Definition 2.1.14 (Disjunctive Normal Form)**.** *A formula F is in disjunctive normal form (DNF), iff*

$$F = (L_{1,1} \wedge \ldots \wedge L_{1,n_1}) \vee \ldots \vee (L_{k,1} \wedge \ldots \wedge L_{k,n_k})$$

where $L_{i,j}$ are literals.

Note that a formula given in CNF or in DNF is in NNF as well. Every propositional logic formula can be transformed into an equivalent formula in CNF/DNF. This can be done by resolving double negations, using De Morgan's laws and using the appropriate distributive rule.

Example 2.1.15. *The formula given in Example 2.1.7*

$$F = \neg(a \vee b \vee (e \wedge \neg f))$$

is not in NNF. Using De Morgan's laws, we can transform F into an equivalent formula F' which is in NNF:

$$F' = \neg a \wedge \neg b \wedge (\neg e \vee f).$$

Note that F' is in CNF as well. Its clause form representation is

$$F'' = \{\{\neg a\}, \{\neg b\}, \{\neg e, f\}\}.$$

The DNF for F is

$$F' = (\neg a \wedge \neg b \wedge \neg e) \vee (\neg a \wedge \neg b \wedge f).$$

2.2 First-Order Logic

In Section 2.2.1 syntax of first-order logic formulae is introduced, followed by Section 2.2.2 which presents semantics of both first-order logic formulae and sets of first-order logic formulae.

2.2.1 Syntax

Definition 2.2.1 (Signature)**.** *A* first-order logic signature *is a tuple* $\Sigma = (\Omega, \Pi)$ *where*

- $\Omega = \{f_1^{a_1}, \ldots, f_n^{a_n}\}$ *is a set of function symbols with arity* $a_i \in \mathbb{N}$*,* $1 \leq i \leq n$*.*
- $\Pi = \{P_1^{a_1}, \ldots, P_n^{a_n}\}$ *is a set of predicate symbols with arity* $a_i \in \mathbb{N}$*,* $1 \leq i \leq n$*.*

Definition 2.2.2 (Syntax of First-Order Logic Terms)**.** *Let* $\Sigma = (\Omega, \Pi)$ *be a first-order logic signature with* $\Omega = \{f_i^k \mid k \in \mathbb{N}, i \in \mathbb{N}^+\}$ *and* X *a countable set of variables. The* set of first-order logic terms *over* X*, denoted by* T_Σ^X*, is the smallest set fulfilling the following:*

- $X \subseteq T_\Sigma^X$*.*
- *If* $t_1, \ldots, t_k \in T_\Sigma^X$ *and* $f_i^k \in \Omega$*, then* $f_i^k(t_1, \ldots, t_k) \in T_\Sigma^X$*.*

Terms of the form f_i^0, meaning with zero arity, are called *constants*. Furthermore, T_Σ denotes the set of ground terms over Σ, meaning the set of all terms in T_Σ^X not containing any variables.

Definition 2.2.3 (Syntax of First-Order Logic Formulae)**.** *Let* $\Sigma = (\Omega, \Pi)$ *be a first-order logic signature,* X *a countable set of variables and* T_Σ^X *the set of first-order logic terms over* X*. The* set of first-order logic formulae F_{fo}^Σ *is the smallest set, fulfilling the follwing:*

- $\bot \in F_{fo}^\Sigma$*.*
- $\top \in F_{fo}^\Sigma$*.*
- *If* $t_1, \ldots, t_k \in T_\Sigma^X$ *and* $P_i^k \in \Pi$*, then* $P_i^k(t_1, \ldots, t_k) \in F_{fo}^\Sigma$*.*
- *If* $x \in X$ *and* $F \in F_{fo}^\Sigma$*, then* $\forall x F \in F_{fo}^\Sigma$ *and* $\exists x F \in F_{fo}^\Sigma$*.*
- *If* $G \in F_{fo}^\Sigma$*, then* $\neg G \in F_{fo}^\Sigma$*.*
- *If* $G, H \in F_{fo}^\Sigma$*, then* $(G \wedge H) \in F_{fo}^\Sigma$ *and* $(G \vee H) \in F_{fo}^\Sigma$*.*

Predicate symbols with zero arity correspond to propositional logic variables. Formulae of the form $P_i^k(t_1, \ldots, t_k)$ are called atomic formulae or atoms. An atom not containing any variables is called *ground atom*. A *literal* is an atom or a negated atom. Like in propositional logic, for first-order logic formulae F and G, we use $F \rightarrow G$ as an abbreviation for $\neg F \vee G$ and $F \leftrightarrow G$ as an abbreviation for $(F \rightarrow G) \wedge (G \rightarrow F)$.

2.2.2 Semantics

Semantics of first-order logic formulae is given with the help of a so-called Σ-interpretation together with a variable valuation. This Σ-interpretation provides a nonempty domain and an interpretation of predicate and function symbols. The domain can be seen as a set of individuals w.r.t. which the respective formula should be interpreted.

Definition 2.2.4 (Σ-interpretation)**.** *Let $\Sigma = (\Omega, \Pi)$ be a first-order logic signature. A Σ-interpretation is a tuple $\mathcal{I} = (U_\mathcal{I}, \Omega_\mathcal{I}, \Pi_\mathcal{I})$, with*

- *$U_\mathcal{I}$ a nonempty set of individuals called domain,*
- *$\Omega_\mathcal{I} = \{f_\mathcal{I} : U_\mathcal{I}^n \to U_\mathcal{I} \mid f^n \in \Omega\}$,*
- *$\Pi_\mathcal{I} = \{P_\mathcal{I} \subseteq U_\mathcal{I}^m \mid P^m \in \Pi\}$.*

The first component of a Σ-interpretation $\mathcal{I} = (U_\mathcal{I}, \Omega_\mathcal{I}, \Pi_\mathcal{I})$ is the domain, often called universe, which can be seen as a set of individuals under consideration. The second component is a set of functions. For each function f occurring in Ω, $\Omega_\mathcal{I}$ contains a function $f_\mathcal{I}$ with same arity as f operating on $U_\mathcal{I}$. The function $f_\mathcal{I}$ corresponds to the interpretation of the function f on the domain $U_\mathcal{I}$. The third component of $\mathcal{I}$ is a set of sets $\Pi_\mathcal{I}$. For each predicate symbol $P \in \Pi$ with arity m, $\Pi_\mathcal{I}$ contains a subset of the m-ary cartesian product of $U_\mathcal{I}$. This subset corresponds to the interpretation of the predicate P on the domain $U_\mathcal{I}$. Since a Σ-interpretation does not interpret variables, it is not possible to interpret first-order formulae using a Σ-interpretation. A valuation function is used, to interpret variables. This valuation function will then be combined with a Σ-interpretation in oder to interpret first-order logic formulae.

Definition 2.2.5 (Valuation Function)**.** *Let $\mathcal{I} = (U_\mathcal{I}, \Omega_\mathcal{I}, \Pi_\mathcal{I})$ be a Σ-interpretation and X a set of variables. A* valuation *is a total function $\beta : X \to U_\mathcal{I}$.*

With the help of a valuation function, it is possible to interpret terms.

Definition 2.2.6 (Valuation of Terms)**.** *Let $\Sigma = (\Omega, \Pi)$ be a first-order logic signature, $\mathcal{I} = (U_\mathcal{I}, \Omega_\mathcal{I}, \Pi_\mathcal{I})$ a Σ-interpretation, X a countable set of variables, $t \in T_\Sigma^X$ a term and $\beta : X \to U_\mathcal{I}$ a valuation. The valuation of term t w.r.t. $\mathcal{I}$ and β is inductively defined by:*

- *$\mathcal{I}_\beta(x) = \beta(x)$ for $x \in X$,*
- *$\mathcal{I}_\beta(f(t_1, \ldots, t_n)) = f_\mathcal{I}(\mathcal{I}_\beta(t_1), \ldots, \mathcal{I}_\beta(t_n))$ for $f^n \in \Omega$.*

Definition 2.2.7 (Semantics of First-Order Logic Formulae)**.** *Let $\Sigma = (\Omega, \Pi)$ be a first-order logic signature, $\mathcal{I} = (U_\mathcal{I}, \Omega_\mathcal{I}, \Pi_\mathcal{I})$ a Σ-interpretation, X a countable set of variables and $\beta : X \to U_\mathcal{I}$ a valuation. The valuation of first-order logic formula by $\mathcal{I}$ w.r.t. β is inductively defined as follows:*

- *$\mathcal{I}_\beta(\bot) = false$,*

- $\mathcal{I}_\beta(\top) = \mathit{true}$,
- $\mathcal{I}_\beta(p(t_1, \ldots t_n)) = \mathit{true}$ *iff* $(I_\beta(t_1), \ldots I_\beta(t_n)) \in P_\mathcal{I}$,
- $\mathcal{I}_\beta(\neg F) = \mathit{true}$ *iff* $\mathcal{I}_\beta(F) = \mathit{false}$,
- $\mathcal{I}_\beta(F_1 \wedge F_2) = \mathit{true}$ *iff* $\mathcal{I}_\beta(F_1) = \mathit{true}$ *and* $\mathcal{I}_\beta(F_2) = \mathit{true}$,
- $\mathcal{I}_\beta(F_1 \vee F_2) = \mathit{true}$ *iff* $\mathcal{I}_\beta(F_1) = \mathit{true}$ *or* $\mathcal{I}_\beta(F_2) = \mathit{true}$,
- $\mathcal{I}_\beta(\forall x F) = \mathit{true}$ *iff* $\mathcal{I}_{\beta[x \mapsto a]}(F) = \mathit{true}$ *for all* $a \in U_\mathcal{I}$,
- $\mathcal{I}_\beta(\exists x F) = \mathit{true}$ *iff* $\mathcal{I}_{\beta[x \mapsto a]}(F) = \mathit{true}$ *for at least one* $a \in U_\mathcal{I}$.

For $x, y \in X$ *and* $a \in U_\mathcal{I}$, $\beta[x \mapsto a]$ *is defined as:*

$$\beta[x \mapsto a](y) := \begin{cases} a & \text{if } x = y, \\ \beta(y) & \text{else.} \end{cases}$$

Given a first-order logic formula together with an interpretation and a variable valuation, Definition 2.2.7 shows how to evaluate the formula in order to find out if it is true in this specific interpretation w.r.t. the variable valuation. With the help of this, we are able to define the notion of a model for a first-order logic formula.

Definition 2.2.8 (Model)**.** *Let* $\Sigma = (\Omega, \Pi)$ *be a first-order logic signature,* $\mathcal{I} = (U_\mathcal{I}, \Omega_\mathcal{I}, \Pi_\mathcal{I})$ *a* Σ*-interpretation,* $\beta : X \to U_\mathcal{I}$ *a variable valuation and* $F \in F_{fo}^\Sigma$ *.* $\mathcal{I}_\beta$ *is a* model *for* F*, denoted by* $\mathcal{I}_\beta \models F$ *iff* $\mathcal{I}_\beta(F) = \mathit{true}$*. Let further* $N \subseteq F_{fo}^\Sigma$ *be a set of first-logic formulae.* $\mathcal{I}_\beta$ *is a model for* N *iff* $\mathcal{I}_\beta \models G$ *for every* $G \in N$*.*

Satisfiability of first-order logic formulae is defined analogous to satisfiability of propositional logic formulae.

Definition 2.2.9 (Satisfiable, Unsatisfiable, Tautology)**.** *A first-order logic formula* F *is called* satisfiable *iff there is a model for* F*. Otherwise* F *is called* unsatisfiable*.* F *is called* tautology *if every interpretation is a model for* F*. A set of first-logic formulae* N *is called satisfiable iff all formulae in* N *are satisfiable. Otherwise* N *is unsatisfiable.*

Note that, contrary to propositional logic, it is not possible to test the satisfiability of a first-order logic formula by checking all possible interpretations. There are infinitely many possibilities to choose a domain for an interpretation and infinitely many ways to interpret predicate and function symbols. Therefore there are infinitely many different interpretations for a first-order logic formula. For details on satisfiability checking of first-order logic formulae we refer the reader to Fitting (1996).

Definition 2.2.10 (Herbrand Interpretation)**.** *Let* $\Sigma = (\Omega, \Pi)$ *be a first-order logic signature with* Ω *containing at least one constant. A* herbrand interpretation *is a* Σ*-interpretation* $\mathcal{I} = (U_\mathcal{I}, \Omega_\mathcal{I}, \Pi_\mathcal{I})$*, with*

- $U_\mathcal{I} = T_\Sigma$ *meaning the universe consists of the set of ground terms over* Σ*.*

- $f_{\mathcal{I}}(s_1, \ldots, s_n) = f(s_1, \ldots, s_n)$ *for* $f^n \in \Omega$ *and* $s_1, \ldots, s_n \in U_{\mathcal{I}}$.

$U_{\mathcal{I}}$ *is called* herbrand universe.

Note that for herbrand interpretations, functions are interpreted as themselves. When constructing a herbrand interpretation, one is only free to choose the interpretation of the predicate symbols.

Definition 2.2.11 (Herbrand Base)**.** *Let* $\Sigma = (\Omega, \Pi)$ *be a first-order logic signature with* Ω *containing at least one constant and* $F \in F^{\Sigma}_{fo}$ *a first-order logic formula. The* herbrand base *of* F *is defined as the following set:*

$$B(F, \Sigma) = \{P(s_1, \ldots, s_n) \mid P^n \in \Pi \text{ and } s_1, \ldots, s_n \in T_{\Sigma}\}.$$

Proposition 2.2.12. *Let* $\Sigma = (\Omega, \Pi)$ *be a first-order logic signature with* Ω *containing at least one constant. Each set of ground atoms* S *identifies exactly one herbrand interpretation* $\mathcal{I} = (U_{\mathcal{I}}, \Omega_{\mathcal{I}}, \Pi_{\mathcal{I}})$*, with* $(s_1, \ldots, s_n) \in P_{\mathcal{I}}$ *iff* $P(s_1, \ldots, s_n) \in S$.

According to Proposition 2.2.12, every set of ground atoms uniquely identifies a herbrand interpretation. Hence, we regard a set of ground atoms and the corresponding herbrand interpretation as equal and do not distinguish among them. With the help of Proposition 2.2.12, it is possible to compare herbrand models using set inclusion leading to the definition of least herbrand models.

2.3 Modal Logic K_n

In Section 2.3.1 syntax of modal logic K_n is introduced followed by Section 2.3.2 which presents semantics of modal logic formulae. Some of the definitions presented in this section were also used by Furbach and Schon (2015).

2.3.1 Syntax

We give the syntax of the modal logic K_n formulae by extending the syntax of propositional logic formuals given in Definition 2.1.1 with a set of unary modal operators $\{\Box_1, \ldots, \Box_n, \Diamond_1, \ldots, \Diamond_n\}$.

Definition 2.3.1 (Syntax of Modal Logic Formulae)**.** *Let* V *be a countable set of propositional logic variables. The set of modal logic* K_n *formulae* $F^V_{\mathsf{K}_n}$ *is the smallest set satisfying the following conditions:*

- $V \subseteq F^V_{\mathsf{K}_n}$,
- *If* $G \in F^V_{\mathsf{K}_n}$*, then* $\neg G$,
- *If* $G \in F^V_{\mathsf{K}_n}$*, then* $\Box_i G \in F^V_{\mathsf{K}_n}$ *and* $\Diamond_i G \in F_{\mathsf{K}_n}$,
- *If* $G, H \in F^V_{\mathsf{K}_n}$*, then* $G \wedge H \in F_{\mathsf{K}_n}$,
- *If* $G, H \in F^V_{\mathsf{K}_n}$*, then* $G \vee H \in F_{\mathsf{K}_n}$.

2.3.2 Semantics

As usual, we give the semantics of the modal logic K_n by introducing Kripke structures.

Definition 2.3.2 (Semantics of Modal Logic Formulae)**.** *Let V be a countable set of propositional logic variables. A* Kripke structure *or* modal logic interpretation *is a tuple $\mathcal{I} = (W, \{R_1, \dots R_n\}, v)$ with*

- *W a nonempty set of worlds,*
- *$\{R_1, \dots, \mathcal{R}_n\}$ a set of reachability relations with $R_i \subseteq W \times W$ for $i \in \{1, \dots, n\}$, and*
- *$v : W \times V \to \{true, false\}$ a valuation function.*

A modal logic interpretation $\mathcal{I} = (W, \{R_1, \dots, R_n\}, v)$ can be seen as transition system. The worlds given in W correspond to the nodes and the reachability relations $\{R_1, \dots R_n\}$ correspond to the edges connecting the nodes. In addition to that, each node is labeled with a set of propositional logic variables. This labeling is done by the valuation function v, such that a world w is labeled with the set of propositional logic variables which are assigned to *true* in w.

Example 2.3.3. *We consider the Kripke structure $\mathcal{I} = (W, \{R_1, R_2\}, v)$ with*

$$
\begin{aligned}
W &= \{w_1, w_2, w_3, w_4\},\\
R_1 &= \{(w_1, w_2), (w_1, w_3), (w_3, w_3)\},\\
R_2 &= \{(w_2, w_2), (w_2, w_4), (w_4, w_3)\},\\
v(w_1, a) &= \{true\},\\
v(w_1, b) &= \{false\},\\
v(w_2, a) &= \{false\},\\
v(w_2, b) &= \{true\},\\
v(w_3, a) &= \{false\},\\
v(w_3, b) &= \{true\},\\
v(w_4, a) &= \{false\}\\
v(w_4, b) &= \{true\}
\end{aligned}
$$

Figure 2.1 depicts a transition system corresponding to this Kripke structure.

In modal logic, different kinds of satisfiability are considered. The first notion is the satisfiability of a modal logic formula in a certain world of a given Kripke structure.

Definition 2.3.4 (Satisfiability of a Modal Logic Formula in a World)**.** *Let $G, H \in F^V_{\mathsf{K}_n}$ be modal logic formulae and $\mathcal{I} = (W, \{R_1, \dots R_n\}, v)$ a Kripke structure. Satisfiability of a modal logic formula G in $\mathcal{I}$ at world $w \in W$, denoted by $\mathcal{I}, w \models G$, is inductively defined as follows:*

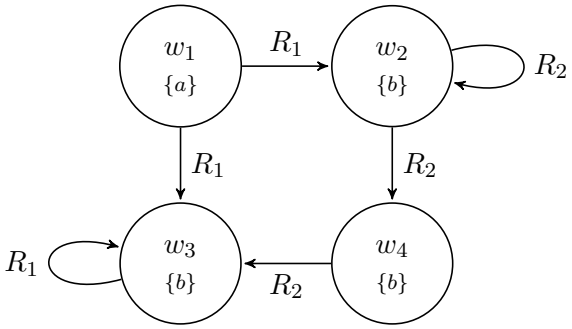


Figure 2.1: Transition System for the Kripke Structure $\mathcal{I}$ given in Example 2.3.3.

- $\mathcal{I}, w \models \top$.
- $\mathcal{I}, w \not\models \bot$.
- $\mathcal{I}, w \models x$ *iff* $v(w, x) = true$, $x \in V$.
- $\mathcal{I}, w \models F \wedge G$ *iff* $\mathcal{I}, w \models F$ *and* $\mathcal{I}, w \models G$.
- $\mathcal{I}, w \models F \vee G$ *iff* $\mathcal{I}, w \models F$ *or* $\mathcal{I}, w \models G$.
- $\mathcal{I}, w \models \Diamond_i F$ *iff there is* $w' \in W$, *with* $R_i(w, w')$ *and* $\mathcal{I}, w' \models F$.
- $\mathcal{I}, w \models \Box i F$ *iff* $\mathcal{I}, w' \models F$ *for all* $w' \in W$ *with* $R_i(w, w')$.

The second notion of satisfiability is the satisfiability of a formula in a Kripke structure.

Definition 2.3.5 (Satisfiability of a Modal Logic Formula in a Kripke Structure)**.** *Let* $F \in F^V_{\mathsf{K}_n}$ *be a modal logic formula and* $\mathcal{I} = (W, \{R_1, \dots R_n\}, v)$ *a Kripke structure.*

- F *is* satisfied *in* $\mathcal{I}$ *if there is a world* $w \in W$ *with* $\mathcal{I}, w \models F$.
- F *is* globally satisfied *in* $\mathcal{I}$, *denoted by* $\mathcal{I} \models F$, *if* F *is satisfied in all worlds* $w \in W$.
- F *is* satisfiable *if there is a Kripke structure where* F *is satisfied in some world.*

Example 2.3.6. *We consider the Kripke structure given in Example 2.3.3 and examine the following modal logic formula in this Kripke structure*

$$F = \Box_2 b \wedge (\Diamond_1 b \vee \Diamond_2 \top)$$

The transition system of the kripke structure consists of 4 different worlds. By considering the transition system given in Figure 2.1, we observe that for all $w \in W$,

$$\mathcal{I}, w \models F$$

Therefore $\mathcal{I} \models F$ *and we can conclude that* F *is satisfiable.*

2.4 Description Logics

Description logics Baader and Nutt (2003) are a family of knowledge representation languages which are more expressive than propositional logic but have better computational properties than first-order logic. Throughout this thesis, different description logics are considered. This is why in Section 2.4.1 both syntax and semantics of the description logics $\mathcal{ALE}$ and $\mathcal{ALC}$ and in Section 2.4.2 the description logic $\mathcal{SHI}$ is introduced. Section 2.4.3 briefly presents reasoning tasks of interest for knowledge bases given in description logics and Section 2.4.4 introduces some equivalences and normal forms. Furthermore, the notion of DL-clauses for description logic knowledge bases given in $\mathcal{SHI}$ is presented in Section 2.4.5.

2.4.1 The Description Logics $\mathcal{ALE}$ and $\mathcal{ALC}$

A knowledge base given in the description logic $\mathcal{ALE}$ or $\mathcal{ALC}$ consists of two parts: a TBox and an ABox. The TBox contains the terminological part of the knowledge. It describes the world using *concepts* and *roles*. Where concepts denote sets of individuals or unary predicates and roles denote binary relations between individuals or binary predicates. The ABox contains the assertional knowledge of the world. It gives information on the individuals occurring in the domain. It specifies the concepts an individual belongs to and how individuals are connected by roles.

Definition 2.4.1 (Description Logic Signature)**.** *A description logic signature is a tuple* $\Sigma = (N_C, N_R, N_I)$ *with*

- N_C *a countable set of atomic concepts,*
- N_R *a countable set of atomic roles, and*
- N_I *a countable set of individuals.*

Definition 2.4.2 (Syntax of $\mathcal{ALE}$ Concepts)**.** *Let* $\Sigma = (N_C, N_R, N_I)$ *be a description logic signature.* $\mathcal{ALE}$ *concepts* C *and* D *are formed according to the following syntax rule:*

$$
\begin{array}{rcll}
C, D & \rightarrow & A \mid & \textit{(atomic concept)} \\
 & & \top \mid & \textit{(top concept)} \\
 & & \bot \mid & \textit{(bottom concept)} \\
 & & \neg A \mid & \textit{(atomic negation)} \\
 & & C \sqcap D \mid & \textit{(intersection)} \\
 & & \exists R.C \mid & \textit{(existential role restriction)} \\
 & & \forall R.C & \textit{(universal role restriction)}
\end{array}
$$

where $A \in N_C$ *is an atomic concept and* $R \in N_R$ *is an atomic role. We use* $C^{\Sigma}_{\mathcal{ALE}}$ *to denote the set of* $\mathcal{ALE}$ *concepts that can be constructed from* Σ*.*

Definition 2.4.3 (Syntax of $\mathcal{ALC}$ Concepts)**.** *Let $\Sigma = (N_C, N_R, N_I)$ be a description logic signature. $\mathcal{ALC}$ concepts C and D are formed according to all syntax rules listed in Definition 2.4.2 together with the following two additional rules*

$$\begin{array}{rcll} C, D & \rightarrow & \neg C \mid & \textit{(negation)} \\ & & C \sqcup D & \textit{(union)} \end{array}$$

We use $C^{\Sigma}_{\mathcal{ALC}}$ to denote the set of $\mathcal{ALC}$ concepts that can be constructed from Σ.

Note that $\mathcal{ALE}$ concepts are $\mathcal{ALC}$ concepts as well. When the description logic under consideration is clear from the context we write C^{Σ} instead of $C^{\Sigma}_{\mathcal{ALE}}$ or $C^{\Sigma}_{\mathcal{ALC}}$ respectively. In the remainder of this section, unless stated otherwise, by concepts we mean $\mathcal{ALC}$ concepts.

Definition 2.4.4 (Role Restriction/Literal Concept/Literal)**.** *A* role restriction *is a concept of the form $\exists R.C$ or $\forall R.C$ with C a concept. A* literal concept *is either $\top$, $\bot$ or of the form A or $\neg A$ for atomic concepts A. Furthermore, a* literal *is either a role restriction or a literal concept.*

Next we define the notion of subconcepts which is a similar to the notion of a subformula in propositional logic as given in Definition 2.1.2.

Definition 2.4.5 (Set of Subconcepts)**.** *Let Σ be a description logic signature C, D, E concepts and R a role.* $\mathsf{sub} : C^{\Sigma} \rightarrow 2^{C^{\Sigma}}$ *is a function mapping a concept C to the smallest set of concepts satisfying the following conditions:*

- $C \in \mathsf{sub}(C)$.
- *If* $\neg D \in \mathsf{sub}(C)$ *then* $D \in \mathsf{sub}(C)$.
- *If* $\exists R.D \in \mathsf{sub}(C)$ *then* $D \in \mathsf{sub}(C)$.
- *If* $\forall R.D \in \mathsf{sub}(C)$ *then* $D \in \mathsf{sub}(C)$.
- *If* $D \sqcap E \in \mathsf{sub}(C)$ *then* $D \in \mathsf{sub}(C)$ *and* $E \in \mathsf{sub}(C)$.
- *If* $D \sqcup E \in \mathsf{sub}(C)$ *then* $D \in \mathsf{sub}(C)$ *and* $E \in \mathsf{sub}(C)$.

For a concept C, $\mathsf{sub}(C)$ *is called the* set of subconcepts *of C.*

Note that we use the same function sub to determine both the set of subformulae of a propositional logic formula and the set of subconcepts of a description logic concept. Since the logic under consideration will always be made clear, this will not lead to confusion.

By subconcepts occurring on the topmost level of a concept C in NNF, we understand each subconcept occurring in C, occurring outside of the scope of a role restriction.

Definition 2.4.6 (Depth of a Concept)**.** *Let Σ be a description logic signature and $C \in C^{\Sigma}$ be a concept.* $\mathsf{depth} : C^{\Sigma} \rightarrow \mathbb{N}$ *is a function mapping a concept C to the maximal depth of nested role restrictions occurring in C and is defined as follows:*

- $\mathsf{depth}(A) = 0$, *for atomic concepts* A.
- $\mathsf{depth}(\neg C) = \mathsf{depth}(C)$.
- $\mathsf{depth}(C \sqcap D) = max(\mathsf{depth}(C), \mathsf{depth}(D))$.
- $\mathsf{depth}(C \sqcup D) = max(\mathsf{depth}(C), \mathsf{depth}(D))$.
- $\mathsf{depth}(\exists R.C) = \mathsf{depth}(C) + 1$.
- $\mathsf{depth}(\forall R.C) = \mathsf{depth}(C) + 1$.

When different normal forms for concepts are considered, it is important to know, if transforming a concept into a certain normal form increases its size.

Definition 2.4.7 (Size of a Concept)**.** *Let* Σ *be a description logic signature,* C *and* D *concepts and* R *a role. The function* $\mathsf{size} : C^{\Sigma} \to \mathbb{N}$ *maps a concept to its size and is defined as follows:*

- $\mathsf{size}(A) = 1$, *for atomic concepts* A *or* $A = \bot$ *or* $A = \top$.
- $\mathsf{size}(\neg C) = \mathsf{size}(C) + 1$.
- $\mathsf{size}(C \sqcap D) = \mathsf{size}(C) + \mathsf{size}(D) + 1$.
- $\mathsf{size}(C \sqcup D) = \mathsf{size}(C) + \mathsf{size}(D) + 1$.
- $\mathsf{size}(\exists R.C) = \mathsf{size}(C) + 2$.
- $\mathsf{size}(\forall R.C) = \mathsf{size}(C) + 2$.

Next semantics of $\mathcal{ALC}$ concepts are presented. Note that, since all $\mathcal{ALE}$ concepts are $\mathcal{ALC}$ concepts as well, the following definition presents the semantics of $\mathcal{ALE}$ concepts as well.

Definition 2.4.8 (Semantics of $\mathcal{ALC}$ Concepts)**.** *An interpretation* $\mathcal{I}$ *is a pair* $\langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$, *where* $\Delta^{\mathcal{I}}$ *is a nonempty set which is the domain of the interpretation and* $\cdot^{\mathcal{I}}$ *is an interpretation function assigning to each atomic concept* A *a set* $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ *and to each atomic role* R *a binary relation* $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. *We extend the interpretation function to complex concepts by the following inductive definitions:*

$$
\begin{aligned}
\top^{\mathcal{I}} &= \Delta^{\mathcal{I}} \\
\bot^{\mathcal{I}} &= \emptyset \\
(\neg C)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(C \sqcup D)^{\mathcal{I}} &= C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
(\exists R.C)^{\mathcal{I}} &= \{a \in \Delta^{\mathcal{I}} \mid \exists b \ (a, b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\} \\
(\forall R.C)^{\mathcal{I}} &= \{a \in \Delta^{\mathcal{I}} \mid \forall b \ (a, b) \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\}
\end{aligned}
$$

Definition 2.4.9 (Model for a Concept)**.** *A concept C is* satisfiable *if there is an interpretation $\mathcal{I}$ with $C^{\mathcal{I}} \neq \emptyset$. We call such an interpretation a* model *for C.*

Definition 2.4.10 (TBox)**.** *Let C and D be concepts. A* terminological axiom *or general concept inclusion (GCI) has the form $C \sqsubseteq D$. A* TBox *consists of a finite set of terminological axioms.*

Definition 2.4.11 (ABox)**.** *Given a set of individuals N_I, an* ABox assertion *has the form $C(a)$ or $R(a, b)$, where C is a concept, R a role and $a, b \in N_I$. An* ABox *is a finite set of ABox assertions.*

Definition 2.4.12 (Description Logic Knowledge Base)**.** *A description logic knowledge base is a tuple $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ with $\mathcal{T}$ a TBox and $\mathcal{A}$ an ABox.*

Sometimes a TBox also contains statements of the form or $C = D$ for two concepts C and D. This statement is an abbreviation for stating that both $C \sqsubseteq D$ and $D \sqsubseteq C$ holds.

Definition 2.4.13 (Satisifiability of a TBox)**.** *A terminological axiom $C \sqsubseteq D$ is satisfied by an interpretation $\mathcal{I}$, if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. A TBox $\mathcal{T}$ is called* satisfiable *if there is an interpretation $\mathcal{I}$ satisfying all its axioms. An interpretation satisfying a TBox $\mathcal{T}$ is called a* model *for $\mathcal{T}$.*

Example 2.4.14. *Consider the following TBox:*

$$\begin{aligned} \mathcal{T} = \{ \quad & \mathit{Father} && \sqsubseteq && \mathit{Man} \sqcap \exists \mathit{hasChild}.\mathit{Human} \\ & \mathit{Mother} && \sqsubseteq && \mathit{Woman} \sqcap \exists \mathit{hasChild}.\mathit{Human} \\ & \mathit{Human} && \sqsubseteq && \mathit{Man} \sqcup \mathit{Woman} \\ & \mathit{Man} \sqcap \mathit{Woman} && \sqsubseteq && \bot \} \end{aligned}$$

This TBox gives some information about families. The first axiom states that a father is a man, who has at least one child. Furthermore it is stated that humans are men or women and that the concepts of women and men are disjoint. Next, we given an interpretation $\mathcal{I}$ for $\mathcal{T}$:

$$\begin{aligned} \Delta^{\mathcal{I}} &= \{b, c, d, e\} \\ \mathit{Father}^{\mathcal{I}} &= \{b\} \\ \mathit{Mother}^{\mathcal{I}} &= \emptyset \\ \mathit{Man}^{\mathcal{I}} &= \{b, d\} \\ \mathit{Woman}^{\mathcal{I}} &= \{c\} \\ \mathit{Human}^{\mathcal{I}} &= \{b, c, d, e\} \\ \mathit{hasChild}^{\mathcal{I}} &= \{(b, d), (b, e)\} \end{aligned}$$

Since the interpretation $\mathcal{I}$ satisfies all axioms given in $\mathcal{T}$, $\mathcal{I}$ is a model for TBox $\mathcal{T}$.

Definition 2.4.15 (Semantics of ABox Assertions, Satisfiability of an ABox)**.** *Let* $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ *be defined as in Definition 2.4.8. We extend the interpretation function such that it maps each individual to a domain element. An ABox assertion* $C(b)$ *is satisfiable if there is an interpretation* $\mathcal{I}$ *with* $b^{\mathcal{I}} \in C^{\mathcal{I}}$*. Furthermore an Abox assertion* $R(b, c)$ *is satisfiable if there is an interpretation* $\mathcal{I}$ *with* $(b^{\mathcal{I}}, c^{\mathcal{I}}) \in R^{\mathcal{I}}$*. We call an interpretation satisfying an ABox assertion a* model *for the ABox assertion. An ABox is called* satisfiable*, if there is an interpretation satisfying all its axioms. An interpretation satisfying an ABox is called a* model *for the ABox.*

Furthermore, an interpretation $\mathcal{I}$ satisfying both $\mathcal{T}$ and $\mathcal{A}$ of a knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ is called a model for the knowledge base. Next we extend the notion of the size of a concept given in Definition 2.4.7 to description logic knowledge bases.

Definition 2.4.16 (Size of a TBox, ABox and KB)**.** *Let* $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ *be a description logic knowledge base and* C*,* D *be concepts. The* size *function introduced in Definition 2.4.7 can be extended such that it can be used to compute the size of terminological axioms, a TBox, ABox assertions, an ABox and a description logic knowledge base by adding the following:*

- $\mathsf{size}(C \sqsubseteq D) = \mathsf{size}(C) + \mathsf{size}(D) + 1$*, for terminological axioms* $C \sqsubseteq D$*.*
- $\mathsf{size}(\mathcal{T}) = \sum_{C \sqsubseteq D \in \mathcal{T}} \mathsf{size}(C \sqsubseteq D)$*.*
- $\mathsf{size}(C(a)) = \mathsf{size}(C) + 1$*, for ABox assertions* $C(a)$*.*
- $\mathsf{size}(R(a, b)) = 3$*, for ABox assertions* $R(a, b)$*.*
- $\mathsf{size}(\mathcal{A}) = \sum_{C(a) \in \mathcal{A}} \mathsf{size}(C(a)) + \sum_{R(a,b) \in \mathcal{A}} \mathsf{size}(R(a, b))$*.*

For a description logic knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ *the size of* $\mathcal{K}$ *is defined as:*

$$\mathsf{size}(\mathcal{K}) = \mathsf{size}(\mathcal{T}) + \mathsf{size}(\mathcal{A})$$

Example 2.4.17. *We consider the following ABox together with the TBox given in Example 2.4.14:*

$$\begin{aligned} \mathcal{A} = \{ & \mathit{Man}(\mathit{norman}), \\ & \mathit{hasChild}(\mathit{norman}, \mathit{liam}), \\ & \mathit{hasChild}(\mathit{norman}, \mathit{julian}) \} \end{aligned}$$

In addition to that the interpretation $\mathcal{I}$ *given in Example 2.4.14 is extended to interpret ABox individuals and assertions:*

$$\begin{aligned} \mathit{norman}^{\mathcal{I}} &= a \\ \mathit{liam}^{\mathcal{I}} &= c \\ \mathit{julian}^{\mathcal{I}} &= d \end{aligned}$$

Since interpretation $\mathcal{I}$ *satisfies* $\mathcal{A}$*, it is a model for* $\mathcal{A}$*. We conclude that* $\mathcal{I}$ *is a model for both* $\mathcal{T}$ *and* $\mathcal{A}$ *and therefore,* $\mathcal{I}$ *is a model for* $\mathcal{K} = (\mathcal{T}, \mathcal{A})$*.*

2.4.2 The Description Logic $\mathcal{SHI}$

The extension of $\mathcal{ALC}$ to include transitively closed primitve roles (Sattler, 1996), role hierarchies and inverse roles is called $\mathcal{SHI}$. The name of this description logic has rather historical reasons. The letter $\mathcal{S}$ is used as an abbreviation for the extension of $\mathcal{ALC}$ with transitively closed primitive roles. The letter $\mathcal{S}$ is used in order to indicate that the resulting logic is related to the multi modal logic $\mathbf{S4}_{(m)}$ Schild (1991) which is modal logic with a reflexive and transitive reachability. The letter $\mathcal{I}$ denotes inverse roles and $\mathcal{H}$ role hierarchies.

In addition to the TBox and the ABox, a knowledge base given in the description logic $\mathcal{SHI}$ contains an RBox. The RBox contains information about the roles used in the knowledge base and gives information about the relation between the different roles.

Definition 2.4.18 (RBox)**.** *For a set of* atomic roles N_R*, the set of* roles *is defined as:*

$$N_R \cup \{R^- \mid R \in N_R\}$$

where R^- denotes the inverse role *corresponding to the atomic role R. A* role inclusion axiom *is an expression of the form $R \sqsubseteq S$, where R and S are atomic or inverse roles. A* transitivity axiom *is of the form $\mathsf{Trans}(S)$ for S an atomic or inverse role. An RBox $\mathcal{R}$ is a finite set of role inclusion axioms and transitivity axioms.*

In order to simplify notation, a function Inv on the set of roles is introduced that computes the inverse of a role, with $\mathsf{Inv}(R) = R^-$ and $\mathsf{Inv}(R^-) = R$.

Since $\mathcal{SHI}$ is the extension of $\mathcal{ALC}$ with transitive roles, role hierarchies and inverse roles, the definition of $\mathcal{SHI}$ concepts and TBoxes are very similar to $\mathcal{ALC}$ concepts and TBoxes. The only difference is, that in $\mathcal{SHI}$, we are allowed to use inverse roles to construct concepts and terminological axioms.

Definition 2.4.19 (Syntax of $\mathcal{SHI}$ Concepts)**.** *Let $\Sigma = (N_C, N_R, N_I)$ be a description logic signature. Complex $\mathcal{SHI}$ concepts C and D are formed according to the following syntax rule:*

$$\begin{array}{rcll} C, D & \rightarrow & A \mid & \textit{(atomic concept)} \\ & & \top \mid & \textit{(top concept)} \\ & & \bot \mid & \textit{(bottom concept)} \\ & & \neg C \mid & \textit{(negation)} \\ & & C \sqcap D \mid & \textit{(intersection)} \\ & & C \sqcup D \mid & \textit{(union)} \\ & & \exists R.C \mid & \textit{(universal role restriction)} \\ & & \forall R.C & \textit{(existential role restriction)} \end{array}$$

where $A \in N_C$ is an atomic concept and $R \in N_R \cup \{R^- \mid R \in N_R\}$. We use $C^{\Sigma}_{\mathcal{SHI}}$ to denote the set of $\mathcal{SHI}$ concepts that can be constructed from Σ.

For a $\mathcal{SHI}$ concept C, the definition of the set of subconcepts of C, denoted by $\mathsf{sub}(C)$, corresponds to Definition 2.4.5.

A $\mathcal{SHI}$ TBox consists of a finite set of terminological axioms $C \sqsubseteq D$ for C, D concepts. Given a set of individuals N_I, a $\mathcal{SHI}$ ABox $\mathcal{A}$ is a finite set of assertions of the form $C(a)$ and $R(a, b)$, with C a concept, R a role and a, b individuals from N_I. A knowledge base $\mathcal{K}$ is a triple $(\mathcal{R}, \mathcal{T}, \mathcal{A})$ with signature $\Sigma = (N_C, N_R, N_I)$.

The size of a $\mathcal{SHI}$ concept corresponds to the size of an $\mathcal{ALC}$ concept as given in Definition 2.4.7. In order to determine the size of a $\mathcal{SHI}$ knowledge base, the definition of the size of an $\mathcal{ALC}$ knowledge base given in Definition 2.4.16 has to be extended to handle RBoxes.

Definition 2.4.20 (Size of an RBox and a KB)**.** *Let R, S be roles. Then the* size *of a role inclusion axiom and an RBox is given as follows:*

- $\mathsf{size}(R \sqsubseteq S) = 3$ *, for role inclusion axioms* $R \sqsubseteq S$*.*
- $\mathsf{size}(\mathsf{Trans}(S)) = 2$*, for transitivity axioms* $\mathsf{Trans}(S)$*.*
- $\mathsf{size}(\mathcal{R}) = \sum_{R \sqsubseteq S \in \mathcal{R}} 3 + \sum_{\mathsf{Trans}(S) \in \mathcal{R}} 2$*.*

For a $\mathcal{SHI}$ knowledge base $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ the size of $\mathcal{K}$ is defined as:

$$\mathsf{size}(\mathcal{K}) = \mathsf{size}(\mathcal{R}) + \mathsf{size}(\mathcal{T}) + \mathsf{size}(\mathcal{A})$$

where $\mathsf{size}(\mathcal{T})$ and $\mathsf{size}(\mathcal{A})$ are defined as in Definition 2.4.16.

Example 2.4.21. *We consider the following RBox together with the TBox given in Example 2.4.14 and the ABox given in Example 2.4.17:*

$$\begin{aligned} \mathcal{R} = \{&\mathsf{Trans}(\mathsf{hasAncestor}), && (2.1)\\ &\mathsf{hasParent} \sqsubseteq \mathsf{hasAncestor}, && (2.2)\\ &\mathsf{hasChild}^- \sqsubseteq \mathsf{hasParent}, && (2.3)\\ &\mathsf{hasParent} \sqsubseteq \mathsf{hasChild}^-\} && (2.4) \end{aligned}$$

The RBox states, that hasAncestor *is a transitive role (2.1),* hasParent *is a sub-role of* hasAncestor *(2.2), and the* hasParent *role is inverse to the* hasChild *role (2.3) and (2.4) .*

Next we define the relation $\sqsubseteq^*$, which gives information on the role hierarchy.

Definition 2.4.22 (Sub-role, Super-role)**.** *For an RBox $\mathcal{R}$, the relation $\sqsubseteq^*$ denotes the reflexive, transitive closure of $\sqsubseteq$ over*

$$\mathcal{R} \cup \{\mathsf{Inv}(R) \sqsubseteq \mathsf{Inv}(S) \mid R \sqsubseteq S \in \mathcal{R}\}$$

A role R is a sub-role (super-role) *of a role S, iff $R \sqsubseteq^* S$ ($S \sqsubseteq^* R$).*

Definition 2.4.23 (Transitive/Simple Role)**.** *Let $\mathcal{R}$ be an RBox. A role R is* transitive *in $\mathcal{R}$ if there is a role S with $S \sqsubseteq^* R$, $R \sqsubseteq^* S$ and $\mathsf{Trans}(S) \in \mathcal{R}$. R is called* simple *if there is no transitive role S with $S \sqsubseteq^* R$.*

Definition 2.4.24 (Semantics of $\mathcal{SHI}$ concepts/knowledge bases)**.** *Let $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ be a $\mathcal{SHI}$ knowledge base with signature $\Sigma = (N_C, N_R, N_I)$. The tuple $\mathcal{I} = (\cdot^{\mathcal{I}}, \Delta^{\mathcal{I}})$ is an* interpretation *for $\mathcal{K}$ iff $\Delta^{\mathcal{I}}$ is a nonempty set and $\cdot^{\mathcal{I}}$ assigns an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ to each individual $a \in N_I$, a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ to each atomic concept A, and a relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ to each atomic role R. $\cdot^{\mathcal{I}}$ then assigns values to more complex concepts and roles as described in Table 2.1. $\mathcal{I}$ is a* model *of $\mathcal{K}$, denoted by $\mathcal{I} \models \mathcal{K}$, if it satisfies all axioms and assertions in $\mathcal{R}$, $\mathcal{T}$ and $\mathcal{A}$ as shown in Table 2.1. A TBox $\mathcal{T}$ is called consistent if there is an interpretation satisfying all axioms in $\mathcal{T}$. A concept C is called*

Interpretation of Concepts and Roles					
$\top^{\mathcal{I}}$	$=$	$\Delta^{\mathcal{I}}$	$(R^-)^{\mathcal{I}}$	$=$	$\{(y,x) \mid (x,y) \in R^{\mathcal{I}}\}$
$\bot^{\mathcal{I}}$	$=$	$\emptyset$	$(\forall R.C)^{\mathcal{I}}$	$=$	$\{x \mid \forall y : (x,y) \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}$
$(\neg C)^{\mathcal{I}}$	$=$	$\Delta^{\mathcal{I}} \backslash C^{\mathcal{I}}$	$(\exists R.C)^{\mathcal{I}}$	$=$	$\{x \mid \exists y : (x,y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
$(C \sqcup D)^{\mathcal{I}}$	$=$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$			
$(C \sqcap D)^{\mathcal{I}}$	$=$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$			

Satisfaction of Axioms in an Interpretation					
$\mathcal{I} \models C \sqsubseteq D$	iff	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$	$\mathcal{I} \models C(a)$	iff	$a^{\mathcal{I}} \in C^{\mathcal{I}}$
$\mathcal{I} \models R \sqsubseteq S$	iff	$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$	$\mathcal{I} \models R(a,b)$	iff	$(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$
$\mathcal{I} \models \mathit{Trans}(R)$	iff	$(R^{\mathcal{I}})^+ \subseteq R^{\mathcal{I}}$			

Table 2.1: Model-Theoretic Semantics of $\mathcal{SHI}$. R^+ is the Transitive Closure of R.

satisfiable w.r.t. *$\mathcal{R}$ and $\mathcal{T}$ iff there exists a model $\mathcal{I}$ of $\mathcal{R}$ and $\mathcal{T}$ with $C^{\mathcal{I}} \neq \emptyset$.*

2.4.3 Reasoning Tasks of Interest

There are several reasoning tasks, which are interesting for description logic knowledge bases. First reasoning tasks for concepts are considered.

Definition 2.4.25 (Satisfiability w.r.t. a TBox)**.** *A concept C is called* satisfiable w.r.t. a TBox $\mathcal{T}$ *if there is a a model $\mathcal{I}$ of $\mathcal{T}$ with $C^{\mathcal{I}} \neq \emptyset$.*

Definition 2.4.26 (Subsumption)**.** *A concept C is* subsumed by a concept D *w.r.t. a TBox $\mathcal{T}$, denoted by $C \sqsubseteq_{\mathcal{T}} D$, if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for all models $\mathcal{I}$ of $\mathcal{T}$.*

Definition 2.4.27 (Equivalence)**.** *A concept C is* equivalent *to a concept D w.r.t. a TBox $\mathcal{T}$, denoted by $C \equiv_{\mathcal{T}} D$, if $C^{\mathcal{I}} = D^{\mathcal{I}}$ for all models $\mathcal{I}$ of $\mathcal{T}$.*

If the TBox is clear from the context or no TBox is used, $C \equiv D$ is written instead of $C \equiv_{\mathcal{T}} D$.

Definition 2.4.28 (Disjointness)**.** *Concepts C and D are called* disjoint *w.r.t. a TBox $\mathcal{T}$ if $C^{\mathcal{I}} \cap D^{\mathcal{I}} = \emptyset$ for all models $\mathcal{I}$ of $\mathcal{T}$.*

As shown in Baader and Nutt (2003), the latter three reasoning tasks can be reduced to checking satisfiability of a concept w.r.t a TBox:

Proposition 2.4.29 (Baader and Nutt (2003)). *Let C, D be concepts and $\mathcal{T}$ be a TBox. Then*

- *$C \sqsubseteq_{\mathcal{T}} D$ iff the concept $C \sqcap \neg D$ is unsatisfiable w.r.t $\mathcal{T}$.*
- *$C \equiv_{\mathcal{T}} D$ iff the two concepts $C \sqcap \neg D$ and $\neg C \sqcap D$ are unsatisfiable w.r.t. $\mathcal{T}$.*
- *C and D are disjoint w.r.t $\mathcal{T}$ iff the concept $C \sqcap D$ is unsatisfiable w.r.t. $\mathcal{T}$.*

Example 2.4.30. *We want to know, if the concepts Father and Human are disjoint w.r.t. the TBox $\mathcal{T}$ given in Example 2.4.14. According to Proposition 2.4.29, Father and Human are disjoint w.r.t. $\mathcal{T}$ iff the concept*

$$\mathit{Father} \sqcap \mathit{Human}$$

is unsatisfiable w.r.t. $\mathcal{T}$. However the model given in Example 2.4.14 is a model for both Father $\sqcap$ Human and $\mathcal{T}$. Therefore we can conclude that Father and Human are not disjoint w.r.t. the TBox $\mathcal{T}$.

Definition 2.4.31 (Consistency). *An ABox $\mathcal{A}$ is called* consistent w.r.t. a TBox $\mathcal{T}$ *if there is an interpretation $\mathcal{I}$, which is a model for both $\mathcal{A}$ and $\mathcal{T}$.*

Definition 2.4.32 (Instance). *An individual b is an* instance *of a concept C w.r.t. $\mathcal{T}$, denoted by $\mathcal{A} \models_{\mathcal{T}} C(b)$, if $b^{\mathcal{I}} \in C^{\mathcal{I}}$ for all models $\mathcal{I}$ of $\mathcal{A}$ and $\mathcal{T}$.*

Definition 2.4.33 (Set of Instances, Instance Retrieval Problem). *For a concept C, a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$, the* set of instances *of C is defined as:*

$$\mathsf{Inst}(C) = \{a \mid \mathcal{A} \models_{\mathcal{T}} C(a)\}$$

The instance retrieval problem *denotes the task of finding all instances of a given concept.*

Definition 2.4.34 (Most Specific Concept). *For an instance b and a set of concepts S, a concept $C \in S$ is called the* most specific concept*, individual b is an instance of if $\mathcal{A} \models_{\mathcal{T}} C(b)$ and C is minimal w.r.t. the subsumption ordering.*

As described in Baader and Nutt (2003), instance checking can be reduced to consistency checking:

Proposition 2.4.35 (Baader and Nutt (2003)). *Let C, D be concepts, $\mathcal{T}$ a TBox and $\mathcal{A}$ an ABox. Then*

$$\mathcal{A} \models_{\mathcal{T}} C(a) \text{ iff } \mathcal{A} \cup \{\neg C(a)\} \text{ is inconsistent w.r.t. } \mathcal{T}.$$

The instance retrieval problem can be reduced to instance checking: For each individual b occurring in the ABox it has to be checked if b is an instance of C. Since instance checking can be reduced to consistency checking, the instance retrieval problem can be reduced to consistency checking as well.

2.4.4 Equivalences and Normal Forms

In the following, several normal forms for description logic concepts will be investigated. To translate a description logic concept into a normal form, it is necessary to perform transformations on concepts. These transformations should work without changing the semantics of the concept, in other words equivalence is supposed to be preserved during the transformation.

The equivalences introduced in Theorem 2.1.10 can be used for concepts given in the description logic $\mathcal{SHI}$ as well. For this, we use a bijective function between concepts and propositional logic formulae.

Definition 2.4.36 (Mapping of Concepts to Propositional Logic Formulae)**.** *Let* $\Sigma = (N_C, N_R, N_I)$ *be a description logic signature.* $\mathsf{prop} : C^{\Sigma}_{\mathcal{SHI}} \to F^{\Sigma'}_{Prop}$ *is a function mapping concepts to propositional logic formulae with*

$$\Sigma' = N_C \cup \{\forall R.C \mid R \in N_R \text{ and } C \in C^{\Sigma}_{\mathcal{SHI}}\} \cup \{\exists R.C \mid R \in N_R \text{ and } C \in C^{\Sigma}_{\mathcal{SHI}}\}$$

defined as follows:

- $\mathsf{prop}(\bot) = \mathit{false}$.
- $\mathsf{prop}(\top) = \mathit{true}$.
- $\mathsf{prop}(B) = b$, *for* B *an atomic concept,* b *a propositional logic variable.*
- $\mathsf{prop}(\neg C) = \neg\mathsf{prop}(C)$, *for concepts* C.
- $\mathsf{prop}(C \sqcap D) = \mathsf{prop}(C) \wedge \mathsf{prop}(D)$, *for concepts* C, D.
- $\mathsf{prop}(C \sqcup D) = \mathsf{prop}(C) \vee \mathsf{prop}(D)$, *for concepts* C, D.
- $\mathsf{prop}(\exists R.C) = \exists R.C$, *with* $\exists R.C$ *a propositional logic variable.*
- $\mathsf{prop}(\forall R.C) = \forall R.C$, *with* $\forall R.C$ *a propositional logic variable.*

Note that computing $\mathsf{prop}(\exists R.(B \sqcap D))$ results in $\exists R.(B \sqcap D)$ which is just a propositional logic variable. This means that prop translates role restrictions to propositional logic variables. However this is not a restriction: if we want to use the propositional logic equivalences for some concept C in the scope of a role restriction $\exists R.C$ ($\forall R.C$), we can easily compute $\mathsf{prop}(C)$ instead of $prop(\exists R.C)$ ($prop(\forall R.C)$).

To facilitate readability, we write propositional logic variables which were produce by the prop function only using lowercase letters and omit the period after the role. For example the propositional logic variable $\exists R.(B \sqcap D)$ will be written as $\exists r(b \sqcap d)$. This allows to distinguish at a first glance between propositional logic variables and role restrictions.

From the way the signature Σ' is constructed, if follows that prop is a bijective function. Therefore prop can be used not only to map concepts to propositional logic formulae but also to map propositional logic formulae to concepts. The prop function can be used in

order to use the propositional logic equivalences introduced in Theorem 2.1.10 as follows: Given a concept C, $\mathsf{prop}(C)$ is computed. Then a subformula of $\mathsf{prop}(C)$ is substituted by some equivalent subformula leading to some propositional logic fomula F. Finally calculating $\mathsf{prop}^{-}(F)$ leads to some concept C' which is equivalent to C.

In the following, when considering transformations of concepts, we sometimes omit the prop mapping and use the equivalences known from propositional logic for concepts.

Theorem 2.4.37. *Let C be a concept. The following equivalences hold:*

$$\neg\exists R.C \equiv \forall R.\neg C$$
$$\neg\forall R.C \equiv \exists R.\neg C$$

Theorem 2.4.37 can be proved using the definition of the semantics of $\mathcal{SHI}$ concepts.

For propositional logic formulae the negation normal form was defined. This normal form can be defined for description logic concepts as well.

Definition 2.4.38 (Negation Normal Form)**.** *A concept C is in* negation normal form (NNF) *iff the negation symbol only occurs in front of atomic concepts.*

Proposition 2.4.39. *For every concept C there is an equivalent concept $\mathsf{nnf}(C)$ which is in negation normal form.*

Every concept can be transformed into NNF by removing double negations, moving the negations as far as possible to the inside of the concept using De Morgan's laws and by using the equivalences given in Theorem 2.4.37. We denote the NNF of a concept C by $\mathsf{nnf}(C)$.

2.4.5 DL-Clauses

Reasoning in description logics is usually done using tableaux calculi. FaCT++ Tsarkov and Horrocks (2006) and Pellet Sirin, Parsia, Grau, Kalyanpur, and Katz (2007) are two examples for highly efficient description logic reasoners which implement tableau-based decision procedures.

The reasoner used in the document at hand is called Hyper and implements a hyper tableau calculus. We will not go into details of this calculus and refer to (Baumgartner, Furbach, and Pelzer, 2007) and (Bender, Pelzer, and Schon, 2013). The calculus implemented in Hyper is closely related to the calculus used in HermiT (Motik, Shearer, and Horrocks, 2007). It is notable that the two reasoners are able to handle quite different logics. Hyper is able to handle full first-order logic with equality and furthermore the description logic $\mathcal{SHIQ}$ which corresponds to the description logic $\mathcal{SHI}$ extended by qualified number restrictions. HermiT on the other had is able to handle OWL ontologies which are much more expressive than $\mathcal{SHIQ}$. In contrast to Hyper, HermiT is not able to handle knowledge bases given in first-order logic. The hyper tableau calculus implemented in HermiT as well as the one implemented in Hyper first transform a description logic knowledge base into so-called *DL-clauses*. The document at hand uses

these DL-clauses in Chapter 4 when introducing an approach for the evolution of ABoxes. Therefore, we will give a description of DL-clauses together with a presentation of how to transform a $\mathcal{SHI}$ knowledge base into DL-clauses.

The transformation of a $\mathcal{SHI}$ knowledge base into DL-clauses is done in three steps: In the first step, transitive roles are removed from the knowledge base. The second step is the normalization of the knowledge base where all TBox axioms and ABox assertions are transformed into a normalized form. In the last step, DL-clauses are constructed from the normalized knowledge base. All three steps will be described in the following.

2.4.5.1 Transformation into DL-Clauses

Elimination of Transitivity Axioms Given a knowledge base in the description logic $\mathcal{SHI}$. In the first step, the knowledge base is transformed into an equisatisfiable $\mathcal{ALCHI}$ knowledge base by eliminating transitive roles. This transformation was introduced by Motik (2006). It is similar to the technique used to transform $\mathcal{SHIQ}$ concepts into $\mathcal{ALCIQ}b$ concepts presented by Tobies (2001) and the encoding of formulae given in the modal logic K4 (K with transitive modality) into formulae of the modal logic K introduced by Tobies (2001). First the notion of the *concept closure* as introduced by Motik (2006) for the description logic $\mathcal{SHIQ}$ is adapted to $\mathcal{SHI}$:

Definition 2.4.40 (Concept Closure (Motik, 2006))**.** *Let $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ be a $\mathcal{SHI}$ knowledge base. The* concept closure *of $\mathcal{K}$, denoted by $\mathsf{clos}(\mathcal{K})$, is defined as the smallest set of concepts satisfying the following conditions:*

- *If $C \sqsubseteq D \in \mathcal{K}$, then $\mathsf{nnf}(\neg C \sqcup D) \in \mathsf{clos}(\mathcal{K})$.*
- *If $C(a) \in \mathcal{A}$, then $\mathsf{nnf}(C) \in \mathsf{clos}(\mathcal{K})$.*
- *If $C \in \mathsf{clos}(\mathcal{K})$ and D is a subconcept of C, then $\mathsf{nnf}(D) \in \mathsf{clos}(\mathcal{K})$.*
- *If $\forall R.C \in \mathsf{clos}(\mathcal{K})$, $S \sqsubseteq^* R$ and $\mathsf{Trans}(S) \in \mathcal{R}$, then $\forall S.C \in \mathsf{clos}(\mathcal{K})$.*

TBox axioms $C \sqsubseteq D$ state that every individual belonging to concept C belongs to concept D as well. In other words there cannot be an individual belonging to C and not belonging to D. Therefore, the axiom can be transformed to $\top \sqsubseteq \neg C \sqcup D$ stating that every individual has to belong to D or to $\neg C$. This is why in Definition 2.4.40 for every TBox axioms $C \sqsubseteq D$ the NNF of the concept $\neg C \sqcup D$ is added to the concept closure.

The first three parts of Definition 2.4.40 can be seen as an extension of the definition of the set of subconcepts given in Definition 2.4.5 to TBoxes and ABoxes: Interpreting the TBox as a set of concepts of the form $\neg C \sqcup D$, the concept closure contains the NNF of all concepts occurring in the TBox or the ABox together with all subconcepts. In the last line of Definition 2.4.40, concepts of the form $\forall R.C$ are considered. For all transitive subroles S of R, $\forall S.C$ is added to the concept closure. This is a very important step, because it enables us later to remove the transitive roles.

Actually the only reason, why we create the concept closure is, that we need the concepts of the form $\forall S.C$ created by the last line of Definition 2.4.40.

The definition of the concept closure is now used in the definition of the Ω-operator introduced by Motik (2006). the Ω operator transforms a $\mathcal{SHI}$ knowledge base $\mathcal{K}$ into an equisatisfiable $\mathcal{ALCHI}$ knowledge base $\Omega(\mathcal{K})$.

Definition 2.4.41 (Ω operator (Motik, 2006))**.** *Let $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ be a $\mathcal{SHI}$ knowledge base. $\Omega(\mathcal{K})$ is the $\mathcal{ALCHI}$ knowledge base constructed as follows:*

- *The RBox of $\Omega(\mathcal{K})$ is obtained from $\mathcal{R}$ by removing all axioms of the form $\mathsf{Trans}(R)$.*
- *The TBox of $\Omega(\mathcal{K})$ is obtained from $\mathcal{T}$ by adding $\forall R.C \sqsubseteq \forall S.(\forall S.C)$, for each concept $\forall R.C \in \mathsf{clos}(\mathcal{K})$ and role S with $S \sqsubseteq^* R$ and $\mathsf{Trans}(S) \in \mathcal{R}$.*
- *The ABox of $\Omega(\mathcal{K})$ corresponds to $\mathcal{A}$.*

According to Motik (2006), the size of $\Omega(\mathcal{K})$ is polynomial in the size of $\mathcal{K}$.

Proposition 2.4.42. *(Motik, 2006) Let $\mathcal{K}$ be a $\mathcal{SHI}$ knowledge base. Then $\mathcal{K}$ and $\Omega(\mathcal{K})$ are equisatisfiable.*

The proof of Proposition 2.4.42 can be found in (Motik, 2006).

Given a knowledge base $\mathcal{K}$, it is possible to use $\Omega(\mathcal{K})$ to answer queries. According to Motik (2006),

$$\Omega(\mathcal{K} \cup \alpha) = \Omega(\mathcal{K}) \cup \{\alpha\}$$

for α of the form $A(b)$, $\neg A(b)$, $S(b, c)$ or $\neg S(b, c)$ with A an atomic concept, S a simple role and b, c individuals.

Example 2.4.43. *The following $\mathcal{SHI}$ knowledge base $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ with*

$$\begin{aligned}
\mathcal{R} &= \{\mathsf{Trans}(S), S \sqsubseteq R\} \\
\mathcal{T} &= \{E \sqsubseteq \forall R.\neg D\} \\
\mathcal{A} &= \{\forall S.C(a_0),\ S(a_0, a_1),\ S(a_1, a_2)\}
\end{aligned}$$

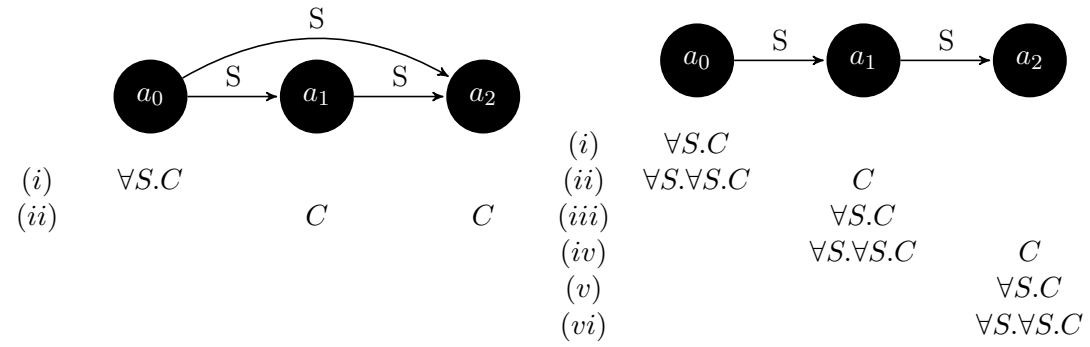
should be transformed into an equisatisfiable $\mathcal{ALCHI}$ knowledge base. For this, first the concept closure of $\mathcal{K}$ is constructed according to Definition 2.4.40.

$$\mathsf{clos}(\mathcal{K}) = \{\neg E \sqcup \forall R.\neg D,\ \neg E,\ \forall R.\neg D,\ E,\ D,\ \neg D,\ \forall S.C,\ C,\ \forall S.\neg D\}$$

In the next step, the transitive role S is removed from $\mathcal{K}$ using Definition 2.4.41 leading to $\Omega(\mathcal{K}) = (\mathcal{R}', \mathcal{T}', \mathcal{A})$ with $\mathcal{R}' = \{S \sqsubseteq R\}$ and

$$\begin{aligned}
\mathcal{T}' = \{&E \sqsubseteq \forall R.\neg D, \\
&\forall S.C \sqsubseteq \forall S.(\forall S.C), \\
&\forall R.\neg D \sqsubseteq \forall S.(\forall S.\neg D), \\
&\forall S.\neg D \sqsubseteq \forall S.(\forall S.\neg D)\}
\end{aligned} \tag{2.5}$$

To illustrate the effect of the Ω-operator transformation presented in Definition 2.4.41, we take a closer look at the original knowledge base $\mathcal{K}$. A kripke structure for the knowledge



(a) ABox of the Knowledge Base $\mathcal{K}$.

(b) ABox of the Knowledge Base $\Omega(\mathcal{K})$.

Figure 2.2: Elimination of Transitive Roles.

base is created successively. Figure 2.2a shows the ABox of $\mathcal{K}$. The individuals occurring in the ABox are represented by black nodes. Furthermore the information given in the ABox is used to label nodes and to create labeled edges between the nodes. For example there is $S(a_0, a_1)$ in the ABox which causes the nodes representing a_0 and a_1 to be connected by an edge labeled with role S. The RBox is used to create the kripke structure as well: the edges connecting node a_0 and a_2 is added because of the transitivity axiom $\mathsf{Trans}(S)$ *in the RBox. Below each node the concepts the corresponding individual belongs to are listed. These concepts are presented in a stepwise manner. The first line shows all concepts for which an instance assertion is contained in the ABox. The second line shows instance assertions which can be deduced from the instance assertions in the first line together with the graph. So since a_0 belongs to concept $\forall S.C$ it is obvious that each S successor of a_0 has to belong to concept C. Hence C is added in the second line below a_1 and a_2.*

Figure 2.2b shows the ABox of $\Omega(\mathcal{K})$. In $\Omega(\mathcal{K})$, the transitivity axiom is removed. Hence there is no edge connecting a_0 and a_2. Again the concepts an individual belongs to are written below the node representing the individual. The first line adds $\forall S.C$ to individual a_0. For the second line we use (2.5) to deduce $\forall S.\forall S.C(a_0)$ and therefore add $\forall S.\forall S.C$ below a_0. Further, since the first line states that a_0 belongs to concept $\forall S.C$ and a_1 is reachable via S from a_0, C is added below a_1. $\forall S.\forall S.C$ given in the second line below a_0 and $S(a_0, a_1)$ implies that we add $\forall C$ to a_1 in the third line. Then again using the TBox axiom (2.5), $\forall S.\forall S.C$ is added to a_1 in the forth line. Commencing in the described way results in the graph shown in Figure 2.2b.

The stepwise construction of the concepts an individual belongs illustrates the idea of the transformation: The TBox axiom $\forall S.C \sqsubseteq \forall S.\forall S.C$ mimics the effect of the transitive role S, ensuring that concept C is added to the same individuals as in Figure 2.2a.

Normalization The second step in the transformation of $\mathcal{SHI}$ knowledge bases into DL-clauses is the normalization. During this step, the TBox axioms and ABox assertions

are transformed into the so-called *normalized form*. The RBox axioms are not changed during the normalization step.

Definition 2.4.44 (Normalization (Motik, Shearer, and Horrocks, 2009))**.** *Let $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ be a $\mathcal{ALCHI}$ knowledge base. A GCI is in* normalized form *if it has the form*

$$\top \sqsubseteq \bigsqcup_{i=1}^{n} C_i$$

where C_i is of the form B, $\forall R.B$, $\exists R.B$ for B a literal concept and R a role. A TBox $\mathcal{T}$ is in normalized form if all its axioms are normalized. An ABox assertion is normalized if it has the form $C(b)$ with C a literal concept or $R(b, c)$ with R an atomic role and b, c individuals. An ABox $\mathcal{A}$ is normalized if all its assertions are normalized and $\mathcal{A}$ is not empty. A knowledge base $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ is normalized if $\mathcal{T}$ and $\mathcal{A}$ are normalized.

During the normalization, auxiliary concepts have to be introduced. When doing this, it is crucial to decide wether to introduce a positive or a negative auxiliary concept. Choosing wisely ensures the DL-clauses obtained in the next step to be as close to horn clauses as possible. To determine if a positive of a negative auxiliary concept is supposed to be introduced, a function called pos is used.

Definition 2.4.45 (pos function (Motik et al., 2007))**.** *Let $\Sigma = (N_C, N_R, N_I)$ be a description logic signature. $\mathsf{pos} : C^{\Sigma}_{\mathcal{SHI}} \rightarrow \{true, false\}$ is a function assigning true or false to concepts which is defined as followed:*

- $\mathsf{pos}(\top) = false$.
- $\mathsf{pos}(\bot) = false$.
- $\mathsf{pos}(A) = true$.
- $\mathsf{pos}(\neg A) = false$
- $\mathsf{pos}(C_1 \sqcap C_2) = \mathsf{pos}(C_1) \wedge \mathsf{pos}(C_2)$.
- $\mathsf{pos}(C_1 \sqcup C_2) = \mathsf{pos}(C_1) \vee \mathsf{pos}(C_2)$.
- $\mathsf{pos}(\forall R.C_1) = \mathsf{pos}(C_1)$.
- $\mathsf{pos}(\exists R.C_1) = true$.

Where A is an atomic concept, C_1 and C_2 are arbitrary concepts.

Definition 2.4.46 (Normalized Knowledge Base (Motik et al., 2009))**.** *Let $\mathcal{K}$ be a $\mathcal{ALCHI}$ knowledge base. The normalized knowledge base $\Delta(\mathcal{K})$ is computed using the functions shown in Table 2.2.*

$$
\begin{array}{rcl}
\Delta(\mathcal{K}) & = & \{\bigcup\limits_{\alpha\in\mathcal{R}} \Delta(\alpha) \cup \\
& & \quad \bigcup\limits_{C_1\sqsubseteq C_2} \Delta(\top \sqsubseteq \mathsf{nnf}(\neg C_1 \sqcup C_2)) \cup \\
& & \quad \{\top(a)\} \cup \bigcup\limits_{\alpha\in\mathcal{A}} \Delta(\alpha)\} \\
\Delta(\top \sqsubseteq \mathbf{C} \sqcup C') & = & \Delta(\top \sqsubseteq \mathbf{C} \sqcup \alpha_{C'}) \\
& & \cup \bigcup\limits_{1\le i\le n} \Delta(\alpha_{C'} \sqsubseteq C_i), \\
& & \text{for } C' = \bigsqcap_{i=1}^{n} C_i \text{ and } n \ge 2 \\
\Delta(\top \sqsubseteq \mathbf{C} \sqcup \forall R.D) & = & \Delta(\top \sqsubseteq \mathbf{C} \sqcup \forall R.\alpha_D) \cup \Delta(\alpha_D \sqsubseteq D) \\
\Delta(\top \sqsubseteq \mathbf{C} \sqcup \exists R.D) & = & \Delta(\top \sqsubseteq \mathbf{C} \sqcup \exists R.\alpha_D) \cup \Delta(\alpha_D \sqsubseteq D) \\
\Delta(D(s)) & = & \{\alpha_D(s)\} \cup \Delta(\alpha_D \sqsubseteq D) \\
\Delta(R^-(s,t)) & = & \{R(t,s)\} \\
\Delta(\beta) & = & \{\beta\}, \text{ for any other axiom}
\end{array}
$$

$$
\alpha_C = \begin{cases} Q_C & \text{if } \mathsf{pos}(C) = \mathit{true} \\ \neg Q_C & \text{if } \mathsf{pos}(C) = \mathit{false} \end{cases} \quad \text{where } Q_C \text{ is a fresh atomic concept unique for } C.
$$

Note: C_i are arbitrary concepts, $\mathbf{C}$ is a possibly empty disjunction of arbitrary concepts, D is not a literal concept, and a is a fresh individual. Since $\sqcup$ is commutative, the C' in $\mathbf{C} \sqcup C'$ is not necessarily the right-most disjunct.

Table 2.2: The Functions used for the Normalization as Introduced by Motik et al. (2007).

Please note that the result of the normalization of a knowledge base is one set containing all normalized axioms from the TBox and the RBox and all normalized ABox assertions. When convenient we will however treat $\Delta(\mathcal{K})$ as a knowledge base $\Delta(\mathcal{K}) = (\mathcal{R}', \mathcal{T}', \mathcal{A}')$ where $\mathcal{R}'$ contains all axioms $R \sqsubseteq S$ with R, S atomic roles, $\mathcal{T}'$ contains all axioms of the form $\top \sqsubseteq \bigsqcup_{i=1}^{n} C_i$ as given in Definition 2.4.44 and $\mathcal{A}'$ contains all assertions of the form $B(b)$ or $R(b,c)$ with B a literal concept, R a role and b, c individuals.

A GCI $C_1 \sqsubseteq C_2$ states that everything that belongs to the concept C_1 belongs to the concept C_2 as well. So the $\sqsubseteq$ sign can be interpreted as an implication. In the first step of the normalization of GCIs, all concepts are moved to the right side of the $\sqsubseteq$. In the next step auxiliary concepts are introduced in order to transform the GCI into the form $\top \sqsubseteq \bigsqcup_{i=1}^{n} C_i$. We illustrate the normalization by an example.

Example 2.4.47. *In Example 2.4.43 we removed the transitive roles from the given $\mathcal{SHI}$ knowledge base $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ leading to the $\mathcal{ALCHI}$ knowledge base $\Omega(\mathcal{K}) = (\mathcal{R}', \mathcal{T}', \mathcal{A})$*

with

$$\begin{aligned}
\mathcal{R}' &= \{S \sqsubseteq R\} \\
\mathcal{T}' &= \{E \sqsubseteq \forall R.\neg D, \\
&\qquad \forall S.C \sqsubseteq \forall S.(\forall S.C), \\
&\qquad \forall R.\neg D \sqsubseteq \forall S.(\forall S.\neg D), \\
&\qquad \forall S.\neg D \sqsubseteq \forall S.(\forall S.\neg D)\} \\
\mathcal{A}' &= \{\forall S.C(a_0),\ S(a_0, a_1),\ S(a_1, a_{2)}\}
\end{aligned}$$

Table 2.2 is used to normalize $\Omega(\mathcal{K})$. The normalization does not transform the RBox. Therefore, the RBox remains unchanged. Normalizing the TBox leads to:

- *Normalization of $E \sqsubseteq \forall R.\neg D$: concept E is moved to the right side in the GCI leading to*

  $$\top \sqsubseteq \neg E \sqcup \forall R.\neg D$$

  Since $\neg D$ is a literal concept, the normalization of this terminological axiom is finished.

- *Normalization of $\forall S.C \sqsubseteq \forall S.(\forall S.C)$: $\forall S.C$ is moved to the right side in the GCI and the right side of the GCI is transformed into negation normal form. This leads to*

  $$\top \sqsubseteq \exists S.\neg C \sqcup \forall S.(\forall S.C)$$

  Since $\forall S.C$ is not a literal concept, an auxiliary concept $\alpha_{\forall S.C}$ has to be introduced, which is either $Q_{\forall S.C}$ or $\neg Q_{\forall S.C}$. We use the pos *function introduced in Definition 2.4.45 in order to find out if the introduced auxiliary concept should be positive or negative. Since*

  $$\mathsf{pos}(\forall S.C) = \mathsf{pos}(C) = \mathit{true}$$

  an auxiliary concept $Q_{\forall S.C}$ is introduced. The resulting two axioms represent the normalization of the original axiom:

  $$\begin{aligned}
  &\top \sqsubseteq \exists S.\neg C \sqcup \forall S.Q_{\forall S.C} \\
  &\top \sqsubseteq \neg Q_{\forall S.C} \sqcup \forall S.C
  \end{aligned}$$

- *Normalization of $\forall R.\neg D \sqsubseteq \forall S.(\forall S.\neg D)$: $\forall R.\neg D$ is moved to the right side in the GCI.*

  $$\top \sqsubseteq \exists R.D \sqcup \forall S.(\forall S.\neg D)$$

  Since $\forall S.\neg D$ is not a literal concept and occurs in the scope of a universal role restriction, an auxiliary concept has to be introduced. Again, the pos *function is used to determine if we have to introduce a positive or a negative auxiliary concept:*

  $$\mathsf{pos}(\forall S.\neg D) = \mathsf{pos}(\neg D) = \mathit{false}$$

Hence the negative auxiliary concept $\neg Q_{\forall S.\neg D}$ is introduced. This leads to the normalized axioms:

$$\top \sqsubseteq \exists R.D \sqcup \forall S.\neg Q_{\forall S.\neg D}$$
$$\top \sqsubseteq Q_{\forall S.\neg D} \sqcup \forall S.\neg D$$

- *Normalization of $\forall S.\neg D \sqsubseteq \forall S.(\forall S.\neg D)$ is similar to normalization of $\forall R.\neg D \sqsubseteq \forall S.(\forall S.\neg D)$. Therefore only the result is presented:*

$$\top \sqsubseteq \exists S.D \sqcup \forall S.\neg Q_{\forall S.\neg D}$$
$$\top \sqsubseteq Q_{\forall S.\neg D} \sqcup \forall S.\neg D$$

In the next step, the ABox is normalized.

- *Normalization of $\forall S.C(a_0)$: Since $\forall S.C$ is not a literal concept, an auxiliary concept, either $Q_{\forall S.C}$ or $\neg Q_{\forall S.C}$, has to be introduced. However during the normalization of the TBox already an auxiliary concept for $\forall S.C$ was introduced. We reuse this concept leading to*

$$Q_{\forall S.C}(a_0)$$
$$\top \sqsubseteq \neg Q_{\forall S.C} \sqcup \forall S.C$$

- *Since all other ABox assertions are already normalized, the normalization of the ABox is finished.*

The resulting normalized knowledge base is: $\Delta(\Omega(\mathcal{K})) = (\mathcal{R}'', \mathcal{T}'', \mathcal{A}'')$ with

$$\begin{aligned}
\mathcal{R}'' &= \{S \sqsubseteq R\} \\
\mathcal{T}'' &= \{\top \sqsubseteq \neg E \sqcup \forall R.\neg D, \\
&\qquad \top \sqsubseteq \exists S.\neg C \sqcup \forall S.Q_{\forall S.C}, \\
&\qquad \top \sqsubseteq \neg Q_{\forall S.C} \sqcup \forall S.C, \\
&\qquad \top \sqsubseteq \exists R.D \sqcup \forall S.\neg Q_{\forall S.\neg D}, \\
&\qquad \top \sqsubseteq Q_{\forall S.\neg D} \sqcup \forall S.\neg D, \\
&\qquad \top \sqsubseteq \exists S.D \sqcup \forall S.\neg Q_{\forall S.\neg D}\} \\
\mathcal{A}'' &= \{Q_{\forall S.C}(a_0),\ S(a_0, a_1),\ S(a_1, a_{2)})\}
\end{aligned}$$

Translation into DL-Clauses In the following, the notion of DL-clauses introduced by Motik et al. (2007) is adapted to the description logic $\mathcal{SHI}$. DL-clauses are well-suited as input format for theorem provers which are based on the hyper tableau calculus to compute models or to decide satisfiability.

Definition 2.4.48 (Atom)**.** *An* atom *is of the form $B(s)$, $R(s,t)$, $\exists R.B(s)$ or $\exists R.\neg B(s)$ for B an atomic concept, R a role and s and t individuals or variables. An atom not containing any variables is called a ground atom.*

Definition 2.4.49 (Syntax of DL-clauses (Motik et al., 2007)). *A* DL-clause *is of the form*

$$U_1 \wedge \ldots \wedge U_m \rightarrow V_1 \vee \ldots \vee V_n$$

with V_i atoms and U_j atoms of the form $B(s)$ or $R(s,t)$ and $m \geq 0$ and $n \geq 0$. If $n = 0$, the right hand side (head) of the DL-clause is denoted by $\bot$. If $m = 0$, the left hand side (body) of the DL-clause is denoted by $\top$. If $n \leq 1$ the DL-clause is called a Horn DL-clause.

Note that the head of a DL-clause is constructed from arbitrary atoms. Since atoms are allowed to be of the form $\exists R.B$, this means that DL-clauses are allowed to contain existential role restrictions. One example for such a DL-clause is

$$C(x) \wedge D(x) \rightarrow \exists R.C(x)$$

This fact constitutes the most obvious difference between the syntax of DL-clauses and first-order logic clauses given as implications. Another difference is the fact that all atoms occurring in DL-clauses are constructed from unary or binary predicates and do not use any function symbols.

Definition 2.4.50 (Set of DL-clauses for a KB (Motik et al., 2007)). *Let $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ be a normalized $\mathcal{ALCHI}$ knowledge base. The set of DL-clauses $\Xi(\mathcal{K})$ for $\mathcal{K}$ is obtained as given in Table 2.3.*

Definition 2.4.51 (Semantics of DL-clauses (Motik et al., 2007)). *Let $\Sigma = (N_C, N_R, N_I)$ be a signature, $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ be a normalized knowledge base over Σ and $\Xi(\mathcal{K})$ the set of DL-clauses for $\mathcal{K}$. Let further*

$$\bigwedge_{i=1}^{m} U_i \rightarrow \bigvee_{j=1}^{n} V_j$$

be a DL-clause in $\Xi(\mathcal{K})$ and N_V a set of variables, disjoint from N_I. Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be an interpretation and $\mu : N_V \rightarrow \Delta^{\mathcal{I}}$ be a variable mapping. Let $b^{\mathcal{I},\mu} = b^{\mathcal{I}}$ for an individual b and $x^{\mathcal{I},\mu} = \mu(x)$ for a variable x. Satisfaction of an atom, a DL-clause, and set of DL-clauses N in $\mathcal{I}$ and μ is defined as follows, where C is a concept, R a role and s, t are variables or individuals:

- $\mathcal{I}, \mu \models C(s)$ *if* $s^{\mathcal{I},\mu} \in C^{\mathcal{I}}$.
- $\mathcal{I}, \mu \models R(s,t)$ *if* $(s^{\mathcal{I},\mu}, t^{\mathcal{I},\mu}) \in R^{\mathcal{I}}$.
- $\mathcal{I}, \mu \models \bigwedge_{i=1}^{m} U_i \rightarrow \bigvee_{j=1}^{n} V_j$ *if* $\mathcal{I}, \mu \models V_j$ *for some* $1 \leq j \leq n$ *whenever* $\mathcal{I}, \mu \models U_i$ *for each* $1 \leq i \leq m$.
- $\mathcal{I} \models \bigwedge_{i=1}^{m} U_i \rightarrow \bigvee_{j=1}^{n} V_j$ *if* $\mathcal{I}, \mu \models \bigwedge_{i=1}^{m} U_i \rightarrow \bigvee_{j=1}^{n} V_j$ *for all mappings* μ.
- $\mathcal{I} \models N$ *if* $\mathcal{I} \models r$ *for each DL-clause* $r \in N$.

An interpretation satisfying a DL-clause (set of DL-clauses) is called *model* for the DL-clause (set of DL-clauses). When convenient, we identify an interpretation with the

$$\Xi(\mathcal{K}) = \{\Big[\bigwedge_{i=1}^{n} \mathsf{lhs}(C_i)\Big] \rightarrow \Big[\bigvee_{i=1}^{n} \mathsf{rhs}(C_i)\Big] \mid \text{for each } \top \sqsubseteq \bigsqcup_{i=1}^{n} C_i \in \mathcal{T}\} \cup$$
$$\{\mathsf{ar}(R,x,y) \rightarrow \mathsf{ar}(S,x,y) \mid \text{for each } (R \sqsubseteq S) \in \mathcal{R}\}$$

$$\mathsf{ar}(R,s,t) = \begin{cases} R(s,t) & \text{if } R \text{ is an atomic role} \\ S(t,s) & \text{if } R \text{ is an inverse role and } R = S^- \end{cases}$$

Note: Whenever $\mathsf{lhs}(C_i)$ or $\mathsf{rhs}(C_i)$ is not defined, it is omitted in the DL-clause.

C	$\mathsf{lhs}(C)$	$\mathsf{rhs}(C)$
A		$A(x)$
$\neg A$	$A(x)$	
$\exists R.A$		$\exists R.A(x)$
$\exists R.\neg A$		$\exists R.\neg A(x)$
$\forall R.A$	$\mathsf{ar}(R,x,y_c)$	$A(y_c)$
$\forall R.\neg A$	$\mathsf{ar}(R,x,y_c) \wedge A(y_c)$	

Table 2.3: Translation of Normalized GCIs into DL-clauses as Introduced by Motik et al. (2007).

set of ground atoms assigned to *true* in the interpretation. This notion allows us to compare interpretations using set operations.

Example 2.4.52. *We transform the result of the normalization performed in Example 2.4.47 into DL-clauses. On the left hand side we show the axioms of the normalized TBox and on the right hand side the corresponding DL-clauses.*

$$\begin{array}{ll} \top \sqsubseteq \neg E \sqcup \forall R.\neg D, & E(x) \wedge R(x,y) \wedge D(y) \rightarrow \bot, \\ \top \sqsubseteq \exists S.\neg C \sqcup \forall S.Q_{\forall S.C}, & S(x,y) \rightarrow Q_{\forall R.C}(y) \vee \exists S.\neg C(x), \\ \top \sqsubseteq \neg Q_{\forall S.C} \sqcup \forall S.C, & Q_{\forall S.C}(x) \wedge S(x,y) \rightarrow C(y), \\ \top \sqsubseteq \exists R.D \sqcup \forall S.\neg Q_{\forall S.\neg D}, & S(x,y) \wedge Q_{\forall S.\neg D}(y) \rightarrow \exists R.D(x), \\ \top \sqsubseteq Q_{\forall S.\neg D} \sqcup \forall S.\neg D, & S(x,y) \wedge D(y) \rightarrow Q_{\forall S.\neg D}(x), \\ \top \sqsubseteq \exists S.D \sqcup \forall S.\neg Q_{\forall S.\neg D} & S(x,y) \wedge Q_{\forall S.\neg D}(y) \rightarrow \exists S.D(x) \end{array}$$

According to Lemma 2 from Motik et al. (2007), DL-clauses can be used to check consistency of a knowledge base.

Lemma 2.4.53. *Let $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ be a normalized $\mathcal{ALCHI}$ knowledge base and $\mathcal{I}$ an interpretation. Then, $\mathcal{I} \models \mathcal{K}$ iff $\mathcal{I} \models \Xi(\mathcal{K})$ and $\mathcal{I} \models \mathcal{A}$.*

2.4.5.2 Properties of DL-Clauses

In this Section, we present some properties of DL-clauses and illustrate benefits of hyper tableau based calculi working with DL-clauses.

Usually tableaux based calculi for description logics handle a TBox

$$\mathcal{T} = \{C_1 \sqsubseteq D_1, \ldots, C_n \sqsubseteq D_n\}$$

by adding the set of concepts

$$\{\neg C_1 \sqcup D_1, \ldots, \neg C_n \sqcup D_n\}$$

to the label of every individual in the tableau (Baader and Nutt, 2003). To process these concepts, the reasoner is forced to nondeterministically choose between $\neg C_i$ and D_i which in the tableau is represented by the creation of two branches. For the first CGI, the reasoner creates two branches. Then, for each following CGI, each branch is extended by two new branches. This clearly leads to exponential behavior. Furthermore, when showing inconsistency of a TBox, every possible branch has to be considered. As soon as a large TBox with many CGIs has to be handled, this leads to extensive branching. The problem arising from that is known as *or-branching* and constitutes a bottleneck for implementations of tableau calcului.

Taking a closer look at the problem shows that this nondeterminism is not always necessary. For example the TBox axiom $A \sqsubseteq B$ is handled by adding $\neg A \sqcup B$ to every individual. However the corresponding DL-clause $A(x) \rightarrow B(x)$ is a Horn DL-clause and shows that the assertion does not necessarily include any nondeterminism. There are techniques like absorption trying to avoid or-branching (Baader and Nutt, 2003), which are able to handle cases like the CGI $A \sqsubseteq B$. When considering more complex CGIs these techniques however are of limited use.

Another way to avoid or-branching is the usage of DL-clauses in combination with a hyper tableau based calculus. As mentioned afore, DL-clauses do not introduce unnecessary nondeterminism. During normalization it is ensured that the resulting DL-clauses are Horn whenever it is possible. For this, special care is taken whenever an auxiliary concept is introduced. $\mathsf{pos}(C)$ is evaluated, whenever during normalization an auxiliary concept has to be introduced as a short-hand for some concept C. If $\mathsf{pos}(C) = \mathit{false}$, C does not require to put something into the head of the DL-clause. This is why a negative auxiliary concept is introduced in this case.

Example 2.4.54. *Let us consider the TBox assertion*

$$\forall R.\neg D \sqsubseteq \forall S.(\forall S.\neg D)$$

given in Example 2.4.47. The first step of the normalization leads to:

$$\top \sqsubseteq \exists R.D \sqcup \forall S.(\forall S.\neg D)$$

Next an auxiliary concept for $\forall S.\neg D$ is introduced. The naive way would be to introduce a positive concept $Q_{\forall S.\neg D}$ leading to:

$$\top \sqsubseteq \exists R.D \sqcup \forall S.Q_{\forall S.\neg D}$$
$$\top \sqsubseteq \neg Q_{\forall S.\neg D} \sqcup \forall S.\neg D$$

This would result in the following two DL-clasues:

$$S(x,y) \rightarrow \exists R.D(x) \vee Q_{\forall S.\neg D}(y)$$
$$Q_{\forall S.\neg D} \wedge S(x,y) \wedge D(y) \rightarrow \bot$$

Note that the first DL-clause is not Horn. However as shown in Example 2.4.47 and 2.4.52 it is possible to transform this TBox assertion into two Horn DL-clauses. The pos *function introduces an negative auxiliary concept* $\neg Q_{\forall S.\neg D}$*. This ensures that the resulting DL-clauses are Horn.*

In Chapter 4, a method to perform revision of ABoxes is presented. This method requires the knowledge base under consideration to be given as set of DL-clauses. During the computation of the ABox revision, it is necessary to construct minimal models and models minimal w.r.t. a set of ground atoms. This is why we present the notion of minimal models and Γ-minimal models for a set of DL-clauses.

Definition 2.4.55 (Minimal Model/Γ-Minimal Model)**.** *Let N be a set of DL-clauses. An interpretation $\mathcal{I}$ is called a minimal model for N iff $\mathcal{I}$ is a model for N and further there is no model $\mathcal{I}'$ for N such that $\mathcal{I}' \subset \mathcal{I}$. Let further Γ be a set of ground atoms. $\mathcal{I}$ is a Γ-minimal model for N iff $\mathcal{I}$ is a model for N and further there is no model $\mathcal{I}'$ for N with $\mathcal{I}' \cap \Gamma \subset \mathcal{I} \cap \Gamma$.*

We close this section by expanding the size function to compute the size of DL-clauses.

Definition 2.4.56 (size Function for DL-Clauses)**.** *Let $C(s)$, $R(s,t)$, $\exists R.C(s)$, D and E be atoms. The function* size *maps a DL-clause to its size and is defined as follows:*

- $\mathsf{size}(C(s)) = 2$.
- $\mathsf{size}(R(s,t) = 3$.
- $\mathsf{size}(\exists R.C(s)) = \mathsf{size}(C) + 3$.
- $\mathsf{size}(\neg D) = \mathsf{size}(D) + 1$.
- $\mathsf{size}(E \wedge D) = \mathsf{size}(E) + \mathsf{size}(D) + 1$.
- $\mathsf{size}(E \vee D) = \mathsf{size}(E) + \mathsf{size}(D) + 1$.
- $\mathsf{size}(E \rightarrow D) = \mathsf{size}(E) + \mathsf{size}(D) + 1$.

2.4.6 Relation to other Logics

2.4.6.1 Propositional Logic

We introduced the description logics $\mathcal{ALE}$, $\mathcal{ALC}$ and $\mathcal{SHI}$. Compared to propositional logic, these logics have the advantage that they are more expressive. In addition to that, these description logics still have the advantage of being decidable.

2.4.6.2 First-Order Logic

The description logic $\mathcal{ALC}$ can be translated into the two variable fragment which is a decidable fragment of first-order logic (Scott, 1962). The two variable fragment consists of first-order formulae with at most two distinct variables. Mappings π_x and π_y translate from $\mathcal{ALC}$ to first-order logic and are inductively defined as given in Table 2.4 (Franz Baader and Diego Calvanese and Deborah L. McGuinness and Daniele Nardi and Peter F. Patel-Schneider, 2003):

$$\begin{aligned}
\pi_x(A) &= A(x), & \pi_y(A) &= A(y),\\
\pi_x(C \sqcap D) &= \pi_x(C) \wedge \pi_x(D), & \pi_y(C \sqcap D) &= \pi_y(C) \wedge \pi_y(D),\\
\pi_x(C \sqcup D) &= \pi_x(C) \vee \pi_x(D), & \pi_y(C \sqcup D) &= \pi_y(C) \vee \pi_y(D),\\
\pi_x(\exists R.C) &= \exists y R(x,y) \wedge \pi_y(C), & \pi_y(\exists R.C) &= \exists x R(y,x) \wedge \pi_x(C),\\
\pi_x(\forall R.C) &= \forall y R(x,y) \rightarrow \pi_y(C), & \pi_y(\forall R.C) &= \forall y R(y,x) \rightarrow \pi_x(C).
\end{aligned}$$

Table 2.4: Translation of $\mathcal{ALC}$ to First-Order Logic.

This translation can be extended to translate transitive roles, inverse roles and role hierarchies as well: For inverse roles, the variables x and y have to be switched. A role hierarchy $R \sqsubseteq S$ can be translated to

$$\forall x, y(R(x,y) \rightarrow S(x,y))$$

An transitivity axiom $\mathsf{Trans}(R)$ can be translated to:

$$\forall x, y, z((R(x,y) \wedge R(y,z)) \rightarrow R(x,z))$$

Note that in this case, the resulting first-order formula is not in the two variable fragment anymore.

In addition to that, we can give a translation π mapping a TBox and an ABox to first-order logic formulae.

$$\begin{aligned}
\pi(\mathcal{T}) &= \bigwedge_{C \sqsubseteq D \in \mathcal{T}} \forall x(\pi_x(C) \rightarrow \pi_x(D))\\
\pi(\mathcal{A}) &= \bigwedge_{R(b,c) \in \mathcal{A}} R(b,c) \wedge \bigwedge_{C(b) \in \mathcal{A}} \pi_{[x|b]}(C)
\end{aligned}$$

where $\pi_{[x|b]}(C)$ denotes the result of substituting b for each occurrence of x in $\pi_x(C)$.

Furthermore, it is noteworthy that, opposed to first-order logic, the description logics $\mathcal{ALE}$, $\mathcal{ALC}$ and $\mathcal{SHI}$ are all decidable.

2.4.6.3 Modal Logic K_n

Schild (1991) shows that there is a strong connection between modal logic and description logic,namely: the description logic $\mathcal{ALC}$ is a notational variant of the modal logic K_n.

Hence it is possible to translate any formula given in the modal logic K_n into an $\mathcal{ALC}$ concept and vice versa. Table 2.5 gives the inductive definition of a mapping ϕ from modal logic K_n formulae to $\mathcal{ALC}$ concepts. Note that in the modal logic K_n we do not have a TBox. Since in K_n there is no way to directly address a world, there is no ABox as well.

Table 2.6 gives the definition of a mapping τ that can be used to translate $\mathcal{ALC}$ concepts into modal logic K_n formulae.

$$\begin{array}{rcl}
\phi(\top) & = & \top \\
\phi(\bot) & = & \bot \\
\phi(a) & = & A \\
\phi(\neg c) & = & \neg\phi(C) \\
\phi(c \wedge d) & = & \phi(C) \sqcap \phi(D) \\
\phi(c \vee d) & = & \varphi(C) \sqcup \phi(D) \\
\phi(\Box_i c) & = & \forall R_i.\phi(C) \\
\phi(\Diamond_i c) & = & \exists R_i.\phi(C)
\end{array}$$

Table 2.5: Translation of Modal Logic K_n Formulae into $\mathcal{ALC}$ Concepts. Where A is an Atomic Concept, C, D are Arbitrary Concepts, R_i is a Role and a, c and d are Variables.

$$\begin{array}{rcl}
\tau(\top) & = & \top \\
\tau(\bot) & = & \bot \\
\tau(A) & = & a \\
\tau(\neg C) & = & \neg\tau(c) \\
\tau(C \sqcap D) & = & \tau(c) \wedge \tau(d) \\
\tau(C \sqcup D) & = & \tau(c) \vee \tau(d) \\
\tau(\forall R_i.(C)) & = & \Box_i\tau(c) \\
\tau(\exists R_i.(C)) & = & \Diamond_i\tau(c)
\end{array}$$

Table 2.6: Translation of $\mathcal{ALC}$ Concepts into Modal Logic K_n Formulae. Where A is an Atomic Concept, C, D are Arbitrary Concepts, R_i is a Role and a, c and d are Variables.

The fact that modal logic K_n and description logics are so closely related offers a way to check satisfiability of a formula F given in the modal logic K_n is to first translate F into the $\mathcal{ALC}$ concept $C := \phi(F)$. Then we can check the satisfiability of $C(a)$ w.r.t. the empty TBox for some individual a. $C(a)$ is satisfiable w.r.t. the empty TBox iff the modal logic formula F is satisfiable.

The following chapters introduce several techniques to process description logic knowledge bases. As a result of the close connection of modal logic $\mathcal{K}_n$ and the description logic $\mathcal{ALC}$, many results presented in the following chapters can be passed on to $\mathcal{K}_n$.

2.5 Hyper Tableau Calculus

In Chapter 4, the hyper tableau calculus will be used to compute change operations on description logic ABoxes. This is why we now briefly introduce some background on this calculus and its implementation Hyper. Since Hyper is only used as a means to construct Γ-minimal models of a set of DL-clauses, details of the hyper tableau calculus are not necessary to understand the approach presented in Capter 4 and we refrain from presenting them.

The hyper tableau calculus is a calculus for first-order logic which is based on clausal normal form tableaux. These hyper tableaux combine many interesting properties of analytic tableaux (De Rijke, 2001) with properties of hyper resolution (Robinson, 1965). As analytic tableaux, hyper tableaux depict the derivation history making it possible to easily retrace the derivation process. In addition to that, hyper tableaux offer a model construction procedure: branches in the hyper tableaux correspond to partial models. Furthermore, finished branches correspond to models. Where a branch is called finished if every clause is redundant in this branch. As in hyper resolution, the hyper tableau calculus allows to resolve away all negative literals of a clause in a single inference step. Further, in the hyper tableau calculus it is ensured that no branch contains repetitions of the same literal. This property is called regularity.

The hyper tableau calculus was first developed for first-order logic reasoning (Baumgartner, Furbach, and Niemelä, 1996) and was extended to handle equality (Baumgartner et al., 2007). Recently, it was extended to handle knowledge bases given in the description logic $\mathcal{SHIQ}$ (Bender et al., 2013). The calculus is implemented in the Hyper theorem prover which is used in various applications. It was used in natural language question answering and commonsense reasoning (Furbach, Glöckner, and Pelzer, 2010; Furbach, Schon, and Stolzenburg, 2015b; Furbach, Schon, Stolzenburg, Weis, and Wirth, 2015c) and in e-learning applications (Baumgartner, Furbach, Groß-Hardt, and Sinner, 2004). Since modal logic and description logic are closely related, the extension of Hyper to handle $\mathcal{SHIQ}$ knowledge bases enables Hyper to handle knowledge bases given in deontic logic as well. Technically speaking, deontic logic corresponds to the modal logic K together with a seriality axiom $\mathsf{D} : \Box P \rightarrow \Diamond P$. This allows the use of Hyper for knowledge bases given in commonsense reasoning applications using deontic logic (Furbach and Schon (2015); Furbach, Gordon, and Schon (2015a)).

3 Knowledge Compilation

It is well known that many reasoning tasks are very much demanding from a computational point of view. For example, for a propositional logic knowledge base KB and a formula F checking if $KB \models F$ in the worst case takes time exponential to the number of distinct variables occurring in KB and F. Checking if a formula is entailed by a knowledge base is only one example for querying a knowledge base. Queries can be seen as operations on a formula which do no change the formula itself but provide information about the formula. In contrast to that, transformations are operations on a formula which change the formula. Examples for queries and transformations are given in Section 3.1.1. When querying a knowledge base, it is possible to split the formula representing the reasoning problem into two parts: the knowledge base and the query. Since the knowledge base is usually not subject to changes, one can assume that the same knowledge base can be used to answer many different queries.

Knowledge compilation is a technique to deal with the intractability of reasoning. The basic idea of knowledge compilation is to divide the reasoning process into two parts: an offline phase and an online phase. During the offline phase, the knowledge base is compiled into a so-called *target language*. This target language does not necessarily have to be human readable but should allow some tractable non-trivial polynomial time queries and transformations (Darwiche and Marquis, 2002). The compilation of the knowledge base itself is often referred to as *precompilation* or *preprocessing*. During the online phase, the knowledge base, now given in the target language, can be queried efficiently for certain types of queries. For example, there are target languages for the compilation of KB, allowing to check if F is entailed in time polynomial in the size of the precompiled version of KB.

In general, the compilation of a knowledge base into a target language can be very expensive. However the precompilation step has to be performed only once. In cases where a large number of queries is posed to the same knowledge base, the compilation of the knowledge base into a target language can be beneficial, since the expense of the precompilation step can be spread over many queries.

Knowledge compilation is a well-known technique for propositional logic. Darwiche and Marquis (2002) present a detailed comparison of different target languages for propositional logic and Cadoli and Donini (1997) provide a survey on knowledge compilation. There are numerous target languages for the precompilation allowing different kinds of queries and transformations to be performed efficiently. When comparing different target languages, three aspects have to be considered:

- The size of the precompiled knowledge base.
- Types of queries that can be answered in time polynomial in the size of the pre-

compiled knowledge base.

- Types of transformations on the precompiled knowledge base which can be performed in time polynomial to the size of the precompiled knowledge base.

The fact that, both queries and transformations, should be polynomial in the size of the precompiled knowledge base illustrates the significance of the size of the precompiled knowledge base.

Please note that knowledge compilation constitutes a problem which is extremely demanding from a computational point of view. For example knowledge compilation in propositional logic is a harder problem than the SAT problem. To solve the SAT problem, only a yes-no answer ist necessary. However to compile a propositional logic formula, it is often necessary to somehow compute all satisfying interpretations or all consequences of the formula. Hence much more has to be computed in order to allow that different queries can be checked efficiently.

The contribution of the thesis at hand is the development of a normal form for concepts and TBoxes given in the descirption logic $\mathcal{ALC}$ which allows certain queries and transformations to be performed efficiently.

We start by introducing knowledge compilation in propositional logic in Section 3.1. In Section 3.1.1, some queries and transformations are introduced for which it is desirable to be able to answer respectively compute them efficiently from a knowledge base given in a target language. Section 3.1.2 introduces several normal forms which can be used as target languages for the knowledge compilation process. Please note that neither the listed queries and transformations nor the presentation of target languages claim to be complete. We selected those target languages which are related to the normal form for the description logic $\mathcal{ALC}$ which is developed in this thesis. Section 3.2 focuses on the task of knowledge compilation for description logic concepts and TBoxes. In Section 3.2, first some normal forms for description logic concepts are developed, then, in Section 3.2.2, the linkless normal form for $\mathcal{ALC}$ concepts is introduced as a target language for $\mathcal{ALC}$ concepts. Furthermore, some interesting properties of the linkless normal form are investigated and presented together with proofs. At the end of Section 3.2.2 it is shown how the linkless normal form can be used as a target language for $\mathcal{ALC}$ TBoxes. Transformation of $\mathcal{ALC}$ TBoxes is implemented and some details on the implementation together with some experimental results are presented. Please note that the focus of this chapter is on the development of the linkless normal form and the investigation of its properties. The implementation only serves for a feasibility study to give an idea on the blowup produced by the compilation and therefore no detailed evaluation is presented. In Section 3.2.4 related work consisting of other target languages for the precompilation process are presented together with some preprocessing techniques besides knowledge compilation which are commonly used in description logic reasoning. The chapter ends with Section 3.3, pointing out avenues interesting for future research.

3.1 Knowledge Compilation in Propositional Logic

In the area of propositional logic, knowledge compilation is a well investigated technique. There are numerous different target languages suitable for different applications. See Darwiche and Marquis (2002) for a detailed presentation of target languages together with queries and transformations which can be carried out efficiently. In Section 3.1.1 desirable properties of target languages are introduced. These properties include queries and transformation which can be performed efficiently for formulae given in a certain target language. The presentation of these queries and transformations does not strive for completeness but only introduces properties which are interesting in the scope of the thesis at hand. Section 3.1.2 introduces some normal forms suitable as a target language namely: linkless formulae, decomposable normal form, prime implicates and prime implicants. Again this presentation of possible target languages does not claim to be complete but rather serves as a basis for the target languages for knowledge compilation of description logic concepts presented in Section 3.2. Furthermore, Section 3.1.2 investigates properties of the introduced target languages.

3.1.1 Methods for Comparision of Target Languages

In propositional logic, there are numerous target languages for the precompilation step. When picking a target language suitable for a specific application, it is advantageous to compare different target languages. Several properties of target languages can be used for comparison. One important aspect is information on a possible blowup caused when transforming a formula into a specific target language. Another aspect is the set of queries, such as consistency checking, which can be answered in polynomial time from formulae given in the target language. Furthermore, it is interesting to know which transformations can be performed in polynomial time on formulae given in the target language. In the following, unless stated otherwise, polytime queries and polytime transformations denote queries or transformations, which can be performed in time polynomial in the size of the precompiled knowledge base. Please note that as soon as the precompilation of the knowledge base caused an exponential blowup, answering queries and performing transformations is polynomial to the exponentially blown up knowledge base. In this case it is neither possible to answer queries efficiently nor possible to perform transformations efficiently. In the following, when considering queries and transformations, we sometimes use the term *polytime* as an abbreviation for *polynomial in the size of the formula*. So we write for example that a normal form allows a polytime consistency check if consistency of all formulae in this normal form can be checked in time polynomial to the size of the formula. Next, different queries and transformations relevant to this thesis are introduced. See (Darwiche, 2001) for an extensive presentation.

3.1.1.1 Queries

A query is an operation on a formula which returns information on the formula without changing it. Types of queries relevant for the thesis at hand are 1. consistency checking,

2. validity checking, 3. clausal entailment checking, 4. determination of the minimal cardinality of a formula and 5. model enumeration. For all these types of queries it is interesting to know the complexity of answering a query for formulae given in a specific target language. In the following, these queries are briefly introduced.

1. *Consistency checking* describes the task to find out if a propositional logic formula F given in a specific target language is satisfiable. If this can be performed in polytime, this query constitutes an interesting property of the respective target language. Even though *satisfiability checking* might appear a more appropriate term for this query, we choose to stick with the term used in the literature.

2. *Validity checking* describes the task to check if a formula given in a specific target language is valid. If the validity check can be performed in polytime, efficient validity checking forms an interesting property of the target language under consideration.

3. *Clausal entailment checking* describes the following problem: Given a propositional logic knowledge base KB in a specific language and a clause C, we want to know if $KB \models C$. Since $KB \models C$ holds iff $KB \wedge \neg C$ is unsatisfiable, target languages for which consistency checking can be performed efficiently are likely to allow an efficient clausal entailment check. If $KB \models C$ can be checked in polytime for KB given in a specific target language, this constitutes a very advantageous property of this target language.

4. *Determination of the minimal cardinality of a formula* describes the task to determine the minimal number of atoms assigned to *false* in a model for a formula. Darwiche (2001) introduces the term *minimum cardinality* for this notion. One example application for this query is model-based diagnosis. Given an observation of a system, the task of model-based diagnosis is to determine the minimal number of faults in the system, which explain the observed behaviour.

   Definition 3.1.1 (Cardinality of a Model/Minimum Cardinality of a Model (Darwiche, 2001))**.** *Let $F \in F^V_{prop}$ be a propositional logic formula and $\mathcal{I}$ an interpretation for F. Then the* cardinality *of $\mathcal{I}$, denoted by $\mathsf{card}(\mathcal{I})$, is the number of atoms set to false in $\mathcal{I}$. The* minimum cardinality *of F, denoted by $\mathsf{mincard}(F)$, is defined as follows:*

$$\mathsf{mincard}(F) = \begin{cases} min(\{\mathsf{card}(\mathcal{I}) \mid \mathcal{I} \models F\}) & \text{if } F \text{ is satisfiable,} \\ \infty & \text{else.} \end{cases}$$

   The minimal cardinality of a formula is the minimal cardinality of all models of the formula.

5. *Model enumeration* describes the task to determine the set of all models of a formula given in a specific target language. Polytime enumeration of models is a desirable but also very challenging property of a target language.

Next, we consider different transformations which can be helpful to compare target languages.

3.1.1.2 Transformations

Opposed to queries on formulae, transformations are operations that use the given input to construct a new formula. From the many transformations that are useful for comparing target languages, we only introduce those that are used in the scope of this thesis, namely 1. combination with logical connectives: $\neg$, $\wedge$ and $\vee$, 2. conditioning, 3. conjoining, 4. projection and 5. minimization of a formula. We refer the reader to (Darwiche and Marquis, 2002) for a more complete and detailed comparison of target languages for knowledge compilation purposes.

In the following, we introduce the transformations mentioned above.

1. *Combination with the logical connectives* $\neg$, $\wedge$ *and* $\vee$ constitutes the most fundamental transformation on formulae. In the context of knowledge compilation, it is interesting to know the complexity of negating a formula given in a specific target language such that the resulting formula is still a formula in the respective target language. If a target language allows to negate formulae in polytime, this target language is denoted to be *closed under negation.* Similar, given two formulae F and G in a target language, it is interesting, if $F \wedge G$ or $F \vee G$ is in the target language. If $F \wedge G$ or $F \vee G$ respectively is in the target language or their target language representation can be computed in time polynomial in the size of F and G, the target language is closed under conjunction or under disjunction respectively. Note that in the context of knowledge compilation, the definition of closure properties differs from the usual definition. When dealing with knowledge compilation it is not only important that the resulting formula is contained in the target language but also that it can be computed in polytime.

2. *Conditioning* is a transformation introduced by Darwiche (1998a, 2001). Intuitively conditioning a formula F on a satisfiable conjunction of literals C is the formula obtained from F by assuming that C is true and simplifying F according to this assumption.

   Definition 3.1.2 (Conditioning (Darwiche, 2001))**.** *Let $F \in F^V_{prop}$ be a propositional logic formula and C be a satisfiable conjunction of literals. The* conditioning *of F on C denoted by $F \mid C$ is the formula obtained by replacing each literal L in F with true if L is a conjunct in C and with false if $\overline{L}$ is a conjunct in C respectively.*

   Note that $F \mid C$ does not contain any atoms occurring in C and $F \wedge C \models F \mid C$. See (Darwiche, 2001) for further properties and applications of the conditioning operator.

3. *Conjoin* is a transformation which is similar to conjunction: Conjoining a formula F and a literal L corresponds to the conjunctive combination of the two, leading to a formula equivalent to $F \wedge L$. Since conjoining handles literals, or satisfiable conjunctions of literals, it is often called *literal conjoin*

Definition 3.1.3 (Conjoin Operator (Darwiche, 2001)). *Let $F \in F^V_{prop}$ be a propositional logic formula and C a satisfiable conjunction of literals.* Conjoining C *to F, denoted by* $\mathsf{conjoin}(F, C)$ *is the formula defined as follows:*

$$\mathsf{conjoin}(F, C) = (F \mid C) \wedge C$$

Obviously $\mathsf{conjoin}(F, C)$ is equivalent to $F \wedge C$.

4. *Projection* of a formula on a set of atoms is another important transformation interesting for knowledge compilation. The basic idea of projecting a formula F on a set of atoms A is the strongest formula in F^A_{prop} implied by F. Projection is dual to the notion of *forgetting* introduced by Lin and Reiter (1994) meaning that projecting a formula F on a set of atoms A yields the same result as forgetting $\overline{A}$ from F.

   Definition 3.1.4 (Projection (Darwiche, 2001)). *Let $F \in F^V_{prop}$ be a propositional logic formula and $A \subseteq V$ be a set of variables. The* projection *of F on A is a propositional logic formula $F' \in F^A_{prop}$ such that for any $G \in F^A_{prop}$, $F' \models G$ iff $F \models G$.*

   Since projection is dual to forgetting, it can be used to remove information from knowledge bases. This is why it comes to use when, depending on the authorization of a user, certain sensitive data is hidden from the user.

5. *Minimization of a formula* describes the following task: Given a propositional logic formula F, the *minimization* of F is a formula whose models are exactly those models of F with minimal cardinality.

   Definition 3.1.5 (Minimization of a Formula (Darwiche, 2001)). *Let $F \in F^V_{prop}$ be a propositional logic formula. A* minimization *of F is a formula F' such that for every interpretation $\mathcal{I}$ over the atoms occurring in F,*

$$\mathcal{I} \models F' \text{ iff } (\mathcal{I} \models F \text{ and } \mathsf{card}(\mathcal{I}) = \mathsf{mincard}(F)).$$

Next, three different target language for knowledge compilation in propositional logic are presented. Furthermore, their respective properties are introduced.

3.1.2 Normal Forms

In this section, three target languages suitable for knowledge compilation in propositional logic are introduced together with their properties. We start with linkless formulae, then introduce the decomposable normal form and end with prime implicants and prime implicates. The section ends with a short comparison of these target languages.

In Section 3.2, the idea of a target language for knowledge compilation in propositional logic will be used to develop a target language for knowledge compilation in the description logic $\mathcal{ALC}$. This is why this presentation of target languages for propositional logic knowledge compilation does not strive for completeness but is restricted to normal forms

related to that specific target language. For a detailed overview on different target languages for knowledge compilation in propositional logic consult (Darwiche and Marquis, 2002) and (Cadoli and Donini, 1997).

3.1.2.1 Linkless Normal Form

The first target language for knowledge compilation in propositional logic we consider are linkless formulae. Informally speaking, a linkless formula is a formula in NNF which does not contain conjunctively combined complementary literals. For example the formula $c \wedge (b \vee \neg c)$ is not linkless since the literals c and $\neg c$ occur conjunctively combined. Linkless formula were introduced by Murray and Rosenthal (1985, 1986) have many interesting properties, which are introduced in the course of this section. Murray and Rosenthal (1985, 1986) represent propositional logic formulae as so-called semantic graphs in order to define the linkless normal form. These semantic graphs are not interesting for the topic of this thesis. Hence we refrain from introducing them and adapt the presentation of the linkless normal form to fit our purposes. This is why the presentation in the following deviates from the presentation by Murray and Rosenthal (1985, 1986).

In order to give a formal definition of this normal form, it is necessary to introduce the notion of paths in a formula as well as the notion of conjunctively combined literals.

Definition 3.1.6 (Path)**.** *Let F_1 and $F_2 \in F^V_{prop}$ be propositional logic formulae.* $\mathsf{paths} : F^V_{prop} \to 2^{F^V_{prop}}$ *is a function mapping a propositional logic formula to a set of sets of literals such that:*

$$\begin{aligned} \mathsf{paths}(F_1) &= \{\{F_1\}\}, \text{ if } F_1 \text{ is a literal or } F_1 = \top \text{ or } F_1 = \bot. \\ \mathsf{paths}(F_1 \vee F_2) &= \mathsf{paths}(F_1) \cup \mathsf{paths}(F_2). \\ \mathsf{paths}(F_1 \wedge F_2) &= \{X \cup Y \mid X \in \mathsf{paths}(F_1) \text{ and } Y \in \mathsf{paths}(F_2)\}. \end{aligned}$$

For a formula F_1, $\mathsf{paths}(F_1)$ is called the set of paths in F_1 *and each element in $\mathsf{paths}(F_1)$ is called a* path in F_1.

Note that for formulae of the form $F = F_1 \wedge F_2$, each path P can be split into the paths P_1 and P_2, where P_1 is path through F_1 and P_2 is a path through F_2 and $P = P_1 \cup P_2$.

In the following, the notion of a path and the formula corresponding to the conjunction of its elements are used interchangeably. This allows us to speak of satisfiable or unsatisfiable paths of a formula.

The following example is based on an example in Murray and Rosenthal (2003).

Example 3.1.7. *Consider the formula*

$$F = \big((\neg a \wedge \neg c) \vee d\big) \wedge \big((c \wedge \neg d) \vee b\big).$$

We construct the set of paths for F using Definition 3.1.6.

$$\begin{aligned} \mathsf{paths}(F) &= \mathsf{paths}\Big(\big((\neg a \wedge \neg c) \vee d\big) \wedge \big((c \wedge \neg d) \vee b\big)\Big) \\ &= \{X \cup Y \mid X \in \mathsf{paths}\big((\neg a \wedge \neg c) \vee d\big) \text{ and } Y \in \mathsf{paths}\big((c \wedge \neg d) \vee b\big)\} \end{aligned}$$

With

$$\begin{aligned}\mathsf{paths}\big((\neg a \wedge \neg c) \vee d\big) &= \mathsf{paths}(\neg a \wedge \neg c) \cup \mathsf{paths}(d)\\ &= \{X \cup Y \mid X \in \mathsf{paths}(\neg a) \text{ and } Y \in \mathsf{paths}(\neg c)\} \cup \mathsf{paths}(d)\\ &= \{X \cup Y \mid X \in \{\{\neg a\}\} \text{ and } Y \in \{\{\neg c\}\}\} \cup \mathsf{paths}(d)\\ &= \{\{\neg a, \neg c\}\} \cup \{\{d\}\}\\ &= \{\{\neg a, \neg c\}, \{d\}\}\end{aligned}$$

$$\begin{aligned}\mathsf{paths}\big((c \wedge \neg d) \vee b\big) &= \mathsf{paths}(c \wedge \neg d) \cup \mathsf{paths}(b)\\ &= \{X \cup Y \mid X \in \mathsf{paths}(c) \text{ and } Y \in \mathsf{paths}(\neg d)\} \cup \mathsf{paths}(b)\\ &= \{X \cup Y \mid X \in \{\{c\}\} \text{ and } Y \in \{\{\neg d\}\}\} \cup \mathsf{paths}(b)\\ &= \{\{c, \neg d\}\} \cup \{\{b\}\}\\ &= \{\{c, \neg d\}, \{b\}\}\end{aligned}$$

this leads to

$$\begin{aligned}\mathsf{paths}(F) &= \{X \cup Y \mid X \in \{\{\neg a, \neg c\}, \{d\}\} \text{ and } Y \in \{\{c, \neg d\}, \{b\}\}\}\\ &= \{\{\neg a, \neg c, c, \neg d\}, \{\neg a, \neg c, b\}, \{d, c, \neg d\}, \{d, b\}\}.\end{aligned}$$

Concluding, we list the four different paths of F:

$$\begin{aligned}P_1 &= \{\neg a, \neg c, c, \neg d\}\\ P_2 &= \{\neg a, \neg c, b\}\\ P_3 &= \{d, c, \neg d\}\\ P_4 &= \{d, b\}\end{aligned}$$

When considering a formula F with a set of paths $\mathsf{paths}(F) = \{P_1, \ldots, P_n\}$, it is easy to see that the formula

$$\bigvee_{i=1}^{n} \bigwedge_{l \in P_i} l$$

is equivalent to F and is in disjunctive normal form.

Example 3.1.8. *Formula F presented in Example 3.1.7 is equivalent to*

$$(\neg a \wedge \neg c \wedge c \wedge \neg d) \vee (\neg a \wedge \neg c \wedge b) \vee (d \wedge c \wedge \neg d) \vee (d \wedge b)$$

which is in disjunctive normal form.

Obviously, a path is inconsistent iff it contains $\bot$ or complementary literals.

Definition 3.1.9 (Conjunctively Combined Literals)**.** *Let $F \in F^V_{prop}$ be a propositional logic formula in NNF. Literals L_1 and L_2 are called* conjunctively combined *in F if there is a path $P \in \mathsf{paths}(F)$ with $L_1 \in P$ and $L_2 \in P$.*

We use paths and conjunctively combined literals to define the notion of a link in a formula as a pair of conjunctively combined complementary literals:

Definition 3.1.10 (Link)**.** *Let $F \in F^V_{prop}$ be a propositional logic formula and $b \in V$ be an atom. The set $\{b, \neg b\}$ is called a* link *in F if there is a path $P \in \mathsf{paths}(F)$ and $\{b, \neg b\} \subseteq P$.*

Clearly, a link constitutes a contradictory part of a formula. Moreover, it is obvious that a formula containing a link in every path is unsatisfiable and a formula containing a path neither containing a link nor $\bot$ is satisfiable.

Example 3.1.11. *Consider formula F given in Example 3.1.7. Path P_3 contains the link $\{d, \neg d\}$. Path P_2, for instance, does not contain a link. Therefore F is satisfiable.*

Definition 3.1.12 (Linkless Normal Form)**.** *Let $F \in F^V_{prop}$ be a propositional logic formula in NNF. F is in* linkless normal form *iff there is no path in F containing a link. If F is in linkless normal form, F is called* linkless*.*

Note that for a set of propositional logic variables V, $\{\{a, \neg a\} \mid a \in V\}$ is the set of all links that can be constructed from variables in V.

Every propositional logic formula can be transformed into an equivalent linkless formula with a method called *path dissolution* (Murray and Rosenthal, 1993, 2003). The result of applying path dissolution to a formula is a linkless formula called *full dissolvent*. The basic idea of path dissolution is to restructure the formula such that all paths containing a link are eliminated.

Path dissolution uses the notion of *path extension* and *path complement* of a formula and a literal. The path extension of a formula F and a literal L is a formula F' such that the paths of F' are exactly those paths of F containing literal L. The path complement of F and L is a formula F' such that the paths in F' are exactly those paths in F not containing literal L.

Murray and Rosenthal (2003) introduced a slightly different nomenclature. They differentiated between conjunctive and disjunctive paths in a formula. The conjunctive paths correspond to the paths introduced in Definition 3.2.2. Since the disjunctive paths are not interesting for the topic of this thesis, we refrain from presenting them making it unnecessary to differentiate between two different types of paths. Furthermore, Murray and Rosenthal (2003) introduce the path complement as *conjunctive path complement* and the path extension as *conjunctive path extension*. Since we only consider one type of paths, we omit the word *conjunctive*.

Definition 3.1.13 (Path Extension/Path Complement)**.** *Let $F \in F^V_{prop}$ be propositional logic formula and L be a literal. The* path extension *of L in F, denoted by $PE(L, F)$, is a formula containing exactly those paths in F which contain L. The* path complement *of L in F, denoted by $PC(L, F)$, is a formula containing exactly those paths in F which do not contain L.*

Note that Definition 3.1.13 does not mention how to construct $PE(L, F)$ and $PC(L, F)$ and we do not present how to construct them. The naive way would be to construct the

disjunction of all respective paths in F. However there are more elaborate methods, producing a far more compact result. Murray and Rosenthal (1993) offer a detailed presentation of ways how to construct both $PE(L, F)$ and $PC(L, F)$ and optimizations of this construction.

Lemma 3.1.14. *Let $F \in F^V_{prop}$ be propositional logic formula and L be a literal occurring in F. Then*

$$F \equiv PE(L, F) \vee PC(L, F).$$

Example 3.1.15. *Consider formulae G_1 and G_2 which are both subformulae of formula F presented in from Example 3.1.7.*

$$G_1 = (\neg a \wedge \neg c) \vee d$$
$$G_2 = (c \wedge \neg d) \vee b$$

Both G_1 and G_2 have two paths:

$$\mathsf{paths}(G_1) = \{\{\neg a, \neg c\}, \{d\}\}$$
$$\mathsf{paths}(G_2) = \{\{c, \neg d\}, \{b\}\}$$

We construct:

$$PE(d, G_1) = d \qquad PE(\neg d, G_2) = c \wedge \neg d$$
$$PC(d, G_1) = \neg a \wedge \neg c \qquad PC(\neg d, G_2) = b$$

As mentioned before, path dissolution can be used to remove links from a formula. Next, a dissolution step for a link $\{L, \overline{L}\}$ in a formula is presented as introduced by Murray and Rosenthal (1993).

Definition 3.1.16 (Dissolvent)**.** *Let $F \in F^V_{prop}$ be a propositional logic formula of the form $F = F_1 \wedge F_2$ containing the link $\{L, \overline{L}\}$ such that $\{L, \overline{L}\}$ is neither a link in F_1 nor in F_2. Let further, w.l.o.g. literal L occur in F_1 and $\overline{L}$ occur in F_2. The* dissolvent *of F and $\{L, \overline{L}\}$ denoted by $\mathsf{diss}(\{L, \overline{L}\}, F)$, is the result of applying the function $\mathsf{diss} : \{\{a, \neg a\} \mid a \in V\} \times F^V_{prop} \to F^V_{prop}$ to $\{L, \overline{L}\}$ and F and is defined as follows:*

$$\begin{aligned}\mathsf{diss}(\{L, \overline{L}\}, F) = &\big(PE(L, F_1) \wedge PC(\overline{L}, F_2)\big) \vee \\ &\big(PC(L, F_1) \wedge PC(\overline{L}, F_2)\big) \vee \\ &\big(PC(L, F_1) \wedge PE(\overline{L}, F_2)\big)\end{aligned}$$

Note that $\mathsf{diss}(\{L, \overline{L}\}, F)$ removes exactly those paths from F containing the link $\{L, \overline{L}\}$. Since these paths are inconsistent, $\mathsf{diss}(\{L, \overline{L}\}, F)$ is equivalent to F. This is stated in the next lemma which was first presented by Murray and Rosenthal (1993) and is adapted to our notation.

Lemma 3.1.17. *Let $F \in F^V_{prop}$ be a propositional logic formula and $\{L, \overline{L}\}$ be a link in F such that $\mathsf{diss}(\{L, \overline{L}\}, F)$ is defined. Then $F \equiv \mathsf{diss}(\{L, \overline{L}\}, F)$.*

By equivalence transformations and with the help of Lemma 3.1.14 the following proposition follows (Murray and Rosenthal, 1993).

Proposition 3.1.18. *Let $\{L, \overline{L}\}$ and F, F_1 and F_2 be defined as in Definition 3.1.16. Then*

$$\mathsf{diss}(\{L, \overline{L}\}, F) \equiv \big(F_1 \wedge PC(\overline{L}, F_2)\big) \vee \big(PC(L, F_1) \wedge PE(\overline{L}, F_2)\big)$$
$$\mathsf{diss}(\{L, \overline{L}\}, F) \equiv \big(F_2 \wedge PC(L, F_1)\big) \vee \big(PC(\overline{L}, F_2) \wedge PE(L, F_1)\big)$$

Proposition 3.1.18 can be used to remove links form a formula. The basic idea to remove a link from a formula F is to first determine the smallest subformula $F_1 \wedge F_2$ of F with F_1 containing the positive part $\{L\}$ of the link and F_2 containing the negative part $\{\overline{L}\}$ of the link. To remove the link from F, it is only necessary to replace $F_1 \wedge F_2$ in F by $\mathsf{diss}(\{L, \overline{L}\}, F_1 \wedge F_2)$. Please note that the order of conjuncts in a subformula is of no importance when choosing $F_1 \wedge F_2$, meaning that if F contains a subformula $G_1 \wedge G_2 \wedge \ldots \wedge G_n$, it is allowed to choose $F_1 = G_i$ and $F_2 = G_j$ for any $1 \leq i, j \leq n$, $i \neq j$.

Example 3.1.19. *Reconsider formulae F given in Example 3.1.7.*

$$F = \big((\neg a \wedge \neg c) \vee d\big) \wedge \big((c \wedge \neg d) \vee b\big)$$

As mentioned previously, F contains the link $\{d, \neg d\}$. Example 3.1.15 shows that F has the form $F = G_1 \wedge G_2$ and presents $PE(d, G_1)$, $PC(d, G_1)$, $PE(\neg d, G_2)$ and $PC(\neg d, G_2)$ which can be used to remove the link $\{d, \neg d\}$ from F. According to Proposition 3.1.18, the dissolvent of F and $\{d, \neg d\}$ is:

$$\begin{aligned}\mathsf{diss}(\{d, \neg d\}, F) &= (G_1 \wedge PC(\neg d, G_2)) \vee (PC(d, G_1) \wedge PE(\neg d, G_2)) \\ &= \Big(\big((\neg a \wedge \neg c) \vee d\big) \wedge b\Big) \vee (\neg a \wedge \neg c \wedge c \wedge \neg d)\end{aligned}$$

Constructing the paths of the result of the dissolution step reveals that the path containing the link $\{d, \neg d\}$ was removed form the formula.

$$\begin{aligned}\mathsf{paths}(\mathsf{diss}(\{d, \neg d\}, F)) = \{&\{\neg a, \neg c, b\}, \\ &\{d, b\}, \\ &\{\neg a, \neg c, c, \neg d\}\}\end{aligned}$$

Note that the link $\{c, \neg c\}$ is still in $diss(\{d, \neg d\}, F)$ and has to be removed in another dissolution step to gain a linkless formula.

As described above, the diss function can be used to remove one link from a formula. The next definition introduces the fulldissolvent function which can be used to remove all links from a given formula. It is obvious that successively removing all links using the diss function leads to a linkless formula. Therefore, one possibility to compute the linkless normal form of a formula is to successively remove all links with the help of the diss function.

Definition 3.1.20 (Full Dissolvent)**.** *Let $F \in F^V_{prop}$ be a propositional logic formula.* $\mathsf{fulldissolvent} : F^V_{prop} \to F^V_{prop}$ *is a function mapping F to an equivalent linkless formula. The result of applying this function to a formula F,* $\mathsf{fulldissolvent}(F)$ *is called* full dissolvent *of F.*

Note that the full dissolvent of a formula and an equivalent linkless formula denote the same term. This is why these two terms will be used interchangeably.

Example 3.1.21. *Consider the formula F presented in Example 3.1.7.*

$$F = \big((\neg a \wedge \neg c) \vee d\big) \wedge \big((c \wedge \neg d) \vee b\big)$$

The full dissolvent for this formula can be constructed from $\mathsf{diss}(\{d, \neg d\}, F)$ *which is computed in Example 3.1.19 by a dissolution step w.r.t the link $\{c, \neg c\}$ leading to*

$$F_1 = \mathsf{fulldissolvent}(F) = ((\neg a \wedge \neg c) \vee d) \wedge b.$$

Please note that transforming a formula into a linkless formula can cause an exponential blowup of the size of the formula. There is an implementation of path dissolution which can be used to transform propositional logic formulae into equivalent linkless formulae. The system is called Dissolver and Murray and Rosenthal (1993) present experimental results.

Properties of Linkless Formulae Firstly, we consider the different queries introduced in Section 3.1.1.1.

1. *Consistency checking:* A linkless formula can be tested for satisfiability in time linear to the size of the formula. When trying to construct a satisfying interpretation for a linkless formula, it is possible to construct satisfying interpretations for all subformulae in the linkless formula independently. Since different conjuncts of a conjunction in a linkless formula are not allowed to contain complementary literals, these satisfying interpretations cannot conflict. This makes it possible to combine the satisfying interpretations of all subformulae into a satisfying interpretation of the whole formula. Note that in case that $\bot$ and $\top$ are not allowed in the formula and are therefore removed by simplifications, the satisfiability of a linkless formula can be checked in constant time.

2. *Validity checking:* Linkless formulae do not allow a polytime validity check. We show this by first introducing the notion of the so-called *simple conjunction property.* A NNF formula satisfies the simple conjunction property if a) the conjuncts of each conjunction are literals and b) do not share variables. DNF formulae naturally satisfies condition a). Condition b) can be easily satisfied by simplifying subformulae of the form $a \wedge \neg a$ or $a \wedge a$. Obviously, every propositional logic formula can be transformed into a DNF satisfying the simple conjunction property. The conjunctions in a DNF formulae satisfying the simple conjunction property do not share variables. Therefore these formulae are linkless. It is well-known that validity checking for DNF formulae is co-NP hard. The existence of a polynomial time

validity check for linkless formulae therefore implies the existence of a polynomial time validity check for DNF formulae satisfying simple conjunction. However this implies $\mathsf{P} = \mathsf{co\text{-}NP}$ leading to $\mathsf{P} = \mathsf{NP}$. We conclude that linkless formulae do not allow a polytime validity check unless $\mathsf{P} = \mathsf{NP}$.

3. *Clausal entailment checking:* For a linkless knowledge base KB and a clause C the clausal entailment $KB \models C$ can be performed in linear time. This follows from the fact that $KB \models C$ holds iff $KB \wedge \neg C$ is satisfiable. Furthermore, $\neg C$ can be transformed into a conjunction of literals with the help of De Morgan's laws. Therefore, conjoining can be used to conjunctively combine KB and $\neg C$ leading to a linkless formula for which a satisfiability test can be performed in linear time. We will see that conjoining can be performed efficiently for linkless formulae.

4. *Other queries:* There are several queries where the linkless normal form does not offer an advantage. Linkless normal form does not allow to determining the minimum cardinality of a formula in polynomial time or polynomial time enumeration of models.

Next, the transformations introduced in Section 3.1.1.2 are considered.

1. *Combination with the logical connectives $\neg$, $\wedge$ and $\vee$:* Naturally, linkless formulae are closed under disjunction. However they are not closed under conjunction and negation unless $\mathsf{P} = \mathsf{NP}$. To see that linkless formulae are not closed under negation, we first recall that as introduced in Section 3.1.1.2, a target language is denoted to be closed under negation if there exists a polytime algorithm that computes $\neg F$ for any arbitrary formula F in the target language and furthermore $\neg F$ is in the target language.

   Let us assume that the literals in each conjunction in a DNF formula do not share variables. This can be achieved in time linear in the size of the DNF formula. With this assumption, every DNF formula is in linkless normal form. We now assume that linkless formulae are closed under negation. Then for every linkless formula F it is possible to compute a linkless formula which is equivalent to $\neg F$ in time polynomial in the size of F. Assume that F is given in DNF. The negation of the DNF formula F is a CNF formula which is obtained from F by applying De Morgan's laws. The assumption that linkless formulae are closed under negation implies that it is possible to transform such a CNF formula into a linkless formula in time polynomial to the size of the CNF formula. This together with the fact that linkless formulae allow to check consistency in linear time implies that $\mathsf{P} = \mathsf{NP}$.

   Darwiche and Marquis (2002) present a proof that DNNF formulae are not closed under conjunction. The proof that linkless formulae are not closed under conjunction can be done analogously to this proof and we refer the reader to Darwiche and Marquis (2002) for details.

2. *Conditioning and conjoining:* According to Rosenthal and Murray (2003), linkless formulae allow polytime conditioning and conjoining. For a given formula F and a

consistent conjunction of literals C, $\mathsf{conjoin}(F, C)$ can be computed in time linear in the size of F.

Example 3.1.22. *Consider the formulae*

$$F_1 = ((\neg a \wedge \neg c) \vee d) \wedge b \text{ and}$$
$$F_2 = b \wedge \neg d.$$

F_1 is linkless and F_2 is a consistent conjunction of literals. We want to conjunctively combine those two formulae. This can be accomplished using the conjoin operator. According to Definition 3.1.3,

$$\begin{aligned}\mathsf{conjoin}(F_1, F_2) &= (F_1 \mid F_2) \wedge F_2 \\ &= \Big(\big(((\neg a \wedge \neg c) \vee d) \wedge b\big) \mid (b \wedge \neg d)\Big) \wedge b \wedge \neg d \\ &= \big(((\neg a \wedge \neg c) \vee \bot) \wedge \top\big) \wedge b \wedge \neg d \\ &= \neg a \wedge \neg c \wedge b \wedge \neg d\end{aligned}$$

which is linkless and equivalent to $F_1 \wedge F_2$.

3. *Projection:* For linkless formulae, projection can be accomplished with the help of a function called project.

   Definition 3.1.23 (project function (Darwiche, 2001))**.** *Let $F, C_i \in F^V_{prop}$, $i \in \mathbb{N}$ be a linkless propositional logic formulae and $A \subseteq V$ a set of propositional logic variables.* $\mathsf{project} : F^V_{prop} \times A \to F^A_{prop}$ *is a function defined as follows:*

$$\mathsf{project}(F, A) = \begin{cases} \top & \text{if } F \text{ is a literal in } F^{V \setminus A}_{prop}, \\ F & \text{if } F \text{ is a literal in } F^A_{prop}, \top or \bot. \end{cases}$$
$$\mathsf{project}(\bigwedge_i C_i, A) = \bigwedge_i \mathsf{project}(C_i, A).$$
$$\mathsf{project}(\bigvee_i C_i, A) = \bigvee_i \mathsf{project}(C_i, A).$$

   Basically the $\mathsf{project}(F, A)$ is the formula resulting from substituting $\top$ for each occurrence of an literal whose atom is in V but not in A in the linkless formula F. According to Murray and Rosenthal (2003), for a linkless formula F and a set of atoms A, the formula $\mathsf{project}(F, A)$ corresponds to the projection of F on the atoms in A, can be computed in time linear in the size of F and is linkless. Hence, linkless formulae are closed under projection.

   Example 3.1.24. *Consider formula F_1 from Example 3.1.21*

$$F_1 = ((\neg a \wedge \neg c) \vee d) \wedge b.$$

and atom set $A = \{b, c\}$*. The projection of* F_1 *on the atoms in* A *is*

$$\begin{aligned}
\mathsf{project}(F_1, A) &= \mathsf{project}(((\neg a \wedge \neg c) \vee d) \wedge b, A) \\
&= \mathsf{project}((\neg a \wedge \neg c) \vee d, A) \wedge \mathsf{project}(b, A) \\
&= (\mathsf{project}(\neg a \wedge \neg c, A) \vee \mathsf{project}(d, A)) \wedge \mathsf{project}(b, A) \\
&= ((\mathsf{project}(\neg a, A) \wedge \mathsf{project}(\neg c, A)) \vee \mathsf{project}(d, A)) \wedge \mathsf{project}(b, A) \\
&= ((\top \wedge \neg c) \vee \top) \wedge b \\
&= b
\end{aligned}$$

4. *Minimization of a formula:* Considering the minimization of a formula described in Section 3.1.1.2, linkless formulae are not helpful. It is not possible to minimize a linkless formula in polynomial time.

3.1.2.2 Decomposable Negation Normal Form

The Decomposable Negation Normal Form (DNNF) introduced by Darwiche (2001) is another target language for knowledge compilation of propositional logic formulae. Like in the case of the linkless formulae, DNNF requires the formula to be in NNF. Furthermore, for a formula to be in DNNF, it is necessary that the conjuncts of each conjunction occurring in the formula do not share atoms.

Definition 3.1.25 (Decomposable Negation Normal Form (DNNF) (Darwiche, 2001))**.** *Let* $F \in F^V_{prop}$ *be a propositional logic formula in NNF.* F *is in* decomposable negation normal form (DNNF) *if* F *satisfies the* decomposability property*: If* $G = G_1 \wedge G_2 \wedge \ldots \wedge G_n \in \mathsf{sub}(F)$ *then* $\mathsf{atoms}(G_i) \neq \mathsf{atoms}(G_j)$ *for* $1 \leq i, j \leq n$*,* $i \neq j$*.*

Example 3.1.26. *The formula* $F = a \wedge b \wedge (e \vee b \vee (a \wedge c))$ *is not in DNNF, since* F *is a conjunction with three different conjuncts and the sets of atoms occurring in the conjuncts are not disjunct as:*

$$\begin{aligned}
\mathsf{atoms}(a) &= \{a\} \\
\mathsf{atoms}(b) &= \{b\} \\
\mathsf{atoms}(e \vee b \vee (a \wedge c))) &= \{e, b, a, c\}
\end{aligned}$$

$$\begin{aligned}
\mathsf{atoms}(a) \cap \mathsf{atoms}(e \vee b \vee (a \wedge c))) &\neq \emptyset \\
\mathsf{atoms}(b) \cap \mathsf{atoms}(e \vee b \vee (a \wedge c))) &\neq \emptyset
\end{aligned}$$

Note that the DNNF can be seen as a special case of linkless formulae: Every formula given in DNNF is linkless as well. On the other hand there are formulae which are linkless but are not in DNNF. Take the formula given in Example 3.1.26 as an example.

As mentioned before, every propositional logic formula can be transformed into a DNF satisfying the simple conjunction property. The conjunctions in a DNF formulae satisfying the simple conjunction property do not share variables. Therefore these formulae

are in DNNF. Hence, every propositional logic formula can be transformed into DNNF by transforming it into a DNF satisfying the simple conjunction property. Since DNNF formulae are not required to be in DNF, this way of transformation does not lead to a succinct DNNF. We will present the basic idea of a different transformations which does not include constructing a DNF. Darwiche (2001) presents details on this transfromation.

Theorem 3.1.27. *(Darwiche, 2001) Let F_1 and F_2 be propositional logic formulae in DNNF and $X = \mathsf{atoms}(F_1) \cap \mathsf{atoms}(F_2)$. Let further C be the set of all consistent conjunctions of literals constructed from atoms in X. Then*

$$F = \bigvee_{C' \in C} \big((F_1 \mid C') \wedge (F_2 \mid C') \wedge C'\big)$$

is in DNNF and F is equivalent to $F_1 \wedge F_2$.

We illustrate how to use Theorem 3.1.27 to transform a formula F into DNNF with the help of an example.

Example 3.1.28. *Consider the propositional logic formula*

$$G_1 = (a \vee \neg e) \wedge (e \vee (b \wedge \neg d)) \wedge (\neg b \vee c).$$

This formula is not in DNNF. We successively transform G_1 into DNNF: First the subformula $(a \vee \neg e) \wedge (e \vee (b \wedge \neg d))$ is considered. Since both $a \vee \neg e$ and $(e \vee (b \wedge \neg d))$ are in DNNF, Theorem 3.1.27 can be used for the DNNF transformation. We set:

$$\begin{aligned} F_1 &= a \vee \neg e \\ F_2 &= e \vee (b \wedge \neg d) \end{aligned}$$

The set of atoms occurring in both F_1 and F_2 is $X = \{e\}$. According to Theorem 3.1.27, $F_1 \wedge F_2$ is equivalent to

$$\begin{aligned} F &= (F_1 \mid e \wedge F_2 \mid e \wedge e) \vee (F_1 \mid \neg e \wedge F_2 \mid \neg e \wedge \neg e) \\ &= (a \wedge \top \wedge e) \vee (\top \wedge (b \wedge \neg d) \wedge \neg e) \\ &= (a \wedge e) \vee (b \wedge \neg d \wedge \neg e) \end{aligned}$$

Since F is equivalent to $F_1 \wedge F_2$, F can be substituted for $F_1 \wedge F_2$ in G_1 leading to:

$$G_2 = ((a \wedge e) \vee (b \wedge \neg d \wedge \neg e)) \wedge (\neg b \vee c)$$

However G_2 is still not in DNNF. Applying Theorem 3.1.27 a second time by setting $F_1 = (a \wedge e) \vee (b \wedge \neg d \wedge \neg e)$ and $F_2 = \neg b \vee c$, leads to the DNNF:

$$G_3 = \big(((a \wedge e) \vee (\neg d \wedge \neg e)) \wedge c \wedge b\big) \vee (a \wedge e \wedge \neg b)$$

which is equivalent to the original formula G_1. Note that this way of transforming a formula into DNNF is nondeterministic. In general there is more than one possibility to

split the formula under consideration into F_1 and F_2. For example in the first step of this example, it would have been possible to set $F_1 = \neg b \vee c$ and $F_2 = a \vee \neg e$. In our example, the order of splitting G does not make a difference. However for larger formulae, the size of the resulting DNNF strongly depends on the order in which the original formula is split.

As shown by Bova, Capelli, Mengel, and Slivovsky (2014), the transformation of a propositional logic formula into DNNF can cause an exponential blowup. Pipatsrisawat and Darwiche (2010) present several algorithms to transform a propositional logic formula into DNNF. An implementation called c2d is available at `http://reasoning.cs.ucla.edu/c2d/` [accessed: 2016, December 22].

Properties of DNNF Formulae Next the different queries of interest for target languages presented in Section 3.1.1.1 are considered and it is investigated, which queries can be answered efficiently for formulae given in DNNF.

1. *Consistency checking:* DNNF formulae allow for consistency checking in time linear in the size of the formula. This is due to the fact that it is possible to construct satisfying interpretations for all subformulae in a DNNF formula independently. These satisfying interpretations cannot conflict, since conjuncts of a conjunction in the DNNF formula are not allowed to share variables. Therefore, the satisfying interpretations can be combined into a satisfying interpretation for the whole formula. This efficient satisfiability test can be performed by the sat predicate defined as follows:

   Definition 3.1.29 (sat Predicate (Darwiche, 2001))**.** *Let $F \in F^V_{prop}$ be a propositional logic formula in DNNF. The sat predicate is defined as follows:*

$$\mathsf{sat}(F) = \begin{cases} \mathit{true} & \mathit{if}\ F\ \mathit{is\ a\ literal\ or}\ \top; \\ \mathit{false} & \mathit{if}\ F = \bot. \end{cases}$$

$$\mathsf{sat}(\bigwedge_i \alpha_i) = \mathit{true}\ \mathit{iff}\ \mathsf{sat}(\alpha_i) = \mathit{true}\ \mathit{for\ all}\ i.$$

$$\mathsf{sat}(\bigvee_i \alpha_i) = \mathit{true}\ \mathit{iff}\ \mathsf{sat}(\alpha_i) = \mathit{true}\ \mathit{for\ some}\ i.$$

   According to Darwiche (2001), for a DNNF formula F, $\mathsf{sat}(F) = \mathit{true}$ iff F is satisfiable. Furthermore, $\mathsf{sat}(F)$ can be computed in time linear in the size of F. This nice property is owed to the fact that the conjuncts of each conjunctions are ensured not to be contradictory. Therefore for each conjunction it is possible to construct a model for each conjunct separately and to combine these models to a model for the conjunction. Note that in case that $\bot$ and $\top$ are not allowed in the formula and are therefore removed by simplifications, the only way a formula F can be unsatisfiable is if $F = \bot$. Hence, in this case, the satisfiability of a DNNF formula can be checked in constant time.

2. *Validity checking:* As mentioned above, formulae in DNF satisfying the simple conjunction property are in DNNF. From this it can be shown that DNNF formulae do not allow a polytime validity check unless $\mathsf{P} = \mathsf{NP}$ by bringing forward the same arguments as in Section 3.1.2.1 in the investigation of validity checking of linkless formulae.

3. *Clausal entailment checking:* For a knowledge base KB given in DNNF and a clause C, clausal entailment $KB \models C$ can be checked in linear time. The reason for this property is the same reason given for linkless formulae in Section 3.1.2.1.

4. *Determination of the minimal cardinality of a formula:* For a DNNF formula F, the minimal cardinality of F, denoted by $\mathsf{mincard}(F)$, can be computed in time linear in the size of F using the mcard function defined as follows:

   Definition 3.1.30 (mcard Function (Darwiche, 1998b))**.** *Let F, $G_i \in F^V_{prop}$, $1 \leq i \leq n$ be propositional logic formulae in DNNF. Function $\mathsf{mcard} : F^V_{prop} \to \mathbb{N}$ is defined as follows:*

$$\mathsf{mcard}(F) = \begin{cases} 0 & \text{if } F \text{ is a positive literal or } \top, \\ 1 & \text{if } F \text{ is a negative literal,} \\ \infty & \text{if } F = \top. \end{cases}$$

$$\mathsf{mcard}(\bigvee_{i=1}^{n} G_i) = min\{\mathsf{mcard}(G_1), \ldots, \mathsf{mcard}(G_n)\}.$$

$$\mathsf{mcard}(\bigwedge_{i=1}^{n} G_i) = \sum_{i=1}^{n} \mathsf{mcard}(G_i).$$

   According to Darwiche (2001), for each DNNF formula F, $\mathsf{mcard}(F) = \mathsf{mincard}(F)$ meaning that the mcard function actually computes the minimum cardinality as defined in Definition 3.1.1.

5. *Model enumeration:* DNNF formulae do not allow for efficiently enumeration of models. However, if the DNNF formula is transformed into the more strict normal form *smooth DNNF*, efficient model enumeration is possible. The basic idea of the smooth DNNF is to add the property that for every disjunction occurring as a subformula, all disjuncts of this disjunction contain the same atoms. In the following definition, when speaking of the set of subformulae $\mathsf{sub}(F)$ of a formula F, we consider $\mathsf{sub}(F)$ to be the set of subformule of F modulo commutativity.

   Definition 3.1.31 (Smooth DNNF (Darwiche, 2000))**.** *Let $F \in F^V_{prop}$ be a propositional logic formula in DNNF. F is in* smooth DNNF *iff for every disjunction $G = G_1 \vee \ldots \vee G_n$ with $G \in \mathsf{sub}(F)$: $\mathsf{atoms}(G) = \mathsf{atoms}(G_i)$ for $1 \leq i \leq n$.*

   Smoothing a DNNF formula, denotes the act of transforming a DNNF formula into a smooth DNNF formula. Given a DNNF formula F of size s with b different atoms, it is possible to compute an equivalent smooth DNNF of size $O(sb)$ in $O(sb)$ time.

The basic idea how to transform a DNNF formula F into an equivalent smooth DNNF formula is the following: For each disjunction

$$G = G_1 \vee \ldots \vee G_k$$

occurring in the formula, k sets of atoms are determined by

$$\overline{A_i} = \mathsf{atoms}(G) \setminus \mathsf{atoms}(G_i).$$

Intuitively, set $\overline{A_i}$ corresponds to the set atoms which are missing in the disjunct G_i. Now each G_i in G is replaced with

$$G_i' = G_i \wedge \bigwedge_{a \in \overline{A_i}} (a \vee \neg a).$$

The resulting formula is equivalent to the original DNNF F and is in smooth DNNF.

Example 3.1.32. *Consider the DNNF constructed in Example 3.1.28:*

$$G_3 = \Big(\big((a \wedge e) \vee (\neg d \wedge \neg e)\big) \wedge c \wedge b\Big) \vee (a \wedge e \wedge \neg b)$$

Firstly, we introduce some abbreviations for subformulae

$$\begin{aligned} G_3 &= \Big(\big((a \wedge e) \vee (\neg d \wedge \neg e)\big) \wedge c \wedge b\Big) \vee (a \wedge e \wedge \neg b) \\ &= \Big(\big(G_1 \vee G_2\big) \wedge c \wedge b\Big) \vee (a \wedge e \wedge \neg b). \end{aligned}$$

Consider the subformula

$$\begin{aligned} G &= G_1 \vee G_2 \\ &= (a \wedge e) \vee (\neg d \wedge \neg e). \end{aligned}$$

G is not smooth, since its first disjunct G_1 contains atoms a and e whereas the second disjunct G_2 contains d and e. In order to smooth this G, we determine

$$\begin{aligned} \mathsf{atoms}(G\ ) &= \{a, d, e\} \\ \mathsf{atoms}(G_1) &= \{a, e\} \\ \mathsf{atoms}(G_2) &= \{d, e\}. \end{aligned}$$

The sets $\overline{A_i}$, containing the atoms lacking in the i-th disjunct are:

$$\begin{aligned} \overline{A_1} &= \{d\} \\ \overline{A_2} &= \{a\} \end{aligned}$$

As stated above,

$$\begin{aligned} G_1' &= a \wedge e \wedge (d \vee \neg d) \\ G_2' &= \neg d \wedge \neg e \wedge (a \vee \neg a) \end{aligned}$$

and G'_i is equivalent to G_i. Substituting G'_i for G_i in G_3 leads to:

$$\begin{aligned} G_4 &= \Big(\big(G'_1 \vee G'_2\big) \wedge c \wedge b\Big) \vee (a \wedge e \wedge \neg b) \\ &= \Big(\big((a \wedge e \wedge (d \vee \neg d)) \vee (\neg d \wedge \neg e \wedge (a \vee \neg a))\big) \wedge c \wedge b\Big) \vee (a \wedge e \wedge \neg b) \end{aligned}$$

The first disjunction in G_4 is now smooth. However the second disjunction is not smooth. Using the same procedure as before, a smooth version of G_4 can be constructed. Leading to

$$G_5 = \Big(\big((a \wedge e \wedge (d \vee \neg d)) \vee (\neg d \wedge \neg e \wedge (a \vee \neg a))\big) \wedge c \wedge b\Big) \vee \Big(a \wedge e \wedge \neg b \wedge (c \vee \neg c) \wedge (d \vee \neg d)\Big)$$

For formulae given in smooth DNNF, it is possible to efficiently enumerate models. The following Definition introduces the models function which can be used to accomplish this task. Please note that in this definition, an interpretation of a propositional logic formula is represented as the set of propositional logic variables assigned to *true* in the interpretation. It is assumed that all variables not occurring in the set representing an interpretation are assigned to *false*.

Definition 3.1.33 (models Function (Darwiche, 2001))**.** *Let F, $G_i \in F^V_{prop}$, $i \in \mathbb{N}$ be a propositional logic formula in smooth DNNF. Function* $\mathsf{models} : F^V_{prop} \rightarrow 2^{2^V}$ *is defined as follows:*

$$\mathsf{models}(F) = \begin{cases} \{\{p\}\} & \text{if } F \text{ is a positive literal } p; \\ \{\{\}\} & \text{if } F \text{ is a negative literal } \neg p \text{ or } F = \top; \\ \{\} & \text{if } F = \bot. \end{cases}$$

$$\mathsf{models}(\bigvee_i G_i) = \bigcup_i \mathsf{models}(G_i).$$

$$\mathsf{models}(\bigwedge_i G_i) = \{\bigcup_i M_i \mid M_i \in \mathsf{models}(G_i)\}.$$

The models function returns exactly the set of models of a given smooth DNNF formula (Darwiche, 1998b) and can therefore be used to enumerate models. Furthermore, the time complexity of the models function is $O(sm^2)$, with s the size of the F and m the number of models for F.

Example 3.1.34. *We consider the smooth DNNF formula G_5 given in Example 3.1.32*

$$\begin{aligned} G_5 &= \Big(\big((a \wedge e \wedge (d \vee \neg d)) \vee (\neg d \wedge \neg e \wedge (a \vee \neg a))\big) \wedge c \wedge b\Big) \vee \\ &\qquad \Big(a \wedge e \wedge \neg b \wedge (c \vee \neg c) \wedge (d \vee \neg d)\Big) \\ &= G_{51} \vee G_{52}. \end{aligned}$$

With

$$\begin{aligned} G_{51} &= \big((a \wedge e \wedge (d \vee \neg d)) \vee (\neg d \wedge \neg e \wedge (a \vee \neg a))\big) \wedge c \wedge b \\ G_{52} &= \Big(a \wedge e \wedge \neg b \wedge (c \vee \neg c) \wedge (d \vee \neg d)\Big). \end{aligned}$$

To determine the set of all models of G_5, the models *function presented in Definition 3.1.33 is used.*

$$\mathsf{models}(G_5) = \mathsf{models}(G_{51} \vee G_{52})$$

G_5 is a disjunction. According to Definition 3.1.33, the models *function can be used to construct the models of each disjunct separately. The set of models of G_5 is then obtained as the union of the constructed sets of models for the disjuncts. In order to determine*

$$\mathsf{models}(G_{51}) = \mathsf{models}\Big(\big((a \wedge e \wedge (d \vee \neg d)) \vee (\neg d \wedge \neg e \wedge (a \vee \neg a))\big) \wedge c \wedge b\Big)$$

we have to compute

$$\begin{aligned}
&\mathsf{models}\big((a \wedge e \wedge (d \vee \neg d)) \vee (\neg d \wedge \neg e \wedge (a \vee \neg a))\big)\\
&\mathsf{models}(c) = \{c\}\\
&\mathsf{models}(b) = \{b\}
\end{aligned}$$

$$\begin{aligned}
&\mathsf{models}\big((a \wedge e \wedge (d \vee \neg d)) \vee (\neg d \wedge \neg e \wedge (a \vee \neg a))\big)\\
&= \mathsf{models}(a \wedge e \wedge (d \vee \neg d)) \cup \mathsf{models}(\neg d \wedge \neg e \wedge (a \vee \neg a))\\
&= \{M_1 \cup M_2 \cup M_3 \mid M_1 \in \mathsf{models}(a) \wedge M_2 \in \mathsf{models}(e) \wedge M_3 \in \mathsf{models}(d \vee \neg d)\} \cup\\
&\quad \{M_1 \cup M_2 \cup M_3 \mid M_1 \in \mathsf{models}(\neg d) \wedge M_2 \in \mathsf{models}(\neg e) \wedge M_3 \in \mathsf{models}(a \vee \neg a)\}\\
&= \{M_1 \cup M_2 \cup M_3 \mid M_1 \in \mathsf{models}(a) \wedge M_2 \in \mathsf{models}(e) \wedge\\
&\qquad\qquad M_3 \in (\mathsf{models}(d) \cup \mathsf{models}(\neg d))\} \cup\\
&\quad \{M_1 \cup M_2 \cup M_3 \mid M_1 \in \mathsf{models}(\neg d) \wedge M_2 \in \mathsf{models}(\neg e) \wedge\\
&\qquad\qquad M_3 \in (\mathsf{models}(a) \cup \mathsf{models}(\neg a))\}\\
&= \{M_1 \cup M_2 \cup M_3 \mid M_1 \in \{a\} \wedge M_2 \in \{e\} \wedge M_3 \in \{\{d\}\} \cup \{\{\neg d\}\}\} \cup\\
&\quad \{M_1 \cup M_2 \cup M_3 \mid M_1 \in \{\neg d\} \wedge M_2 \in \{\neg e\} \wedge M_3 \in \{\{a\}\} \cup \{\{\neg a\}\}\}\\
&= \{M_1 \cup M_2 \cup M_3 \mid M_1 \in \{a\} \wedge M_2 \in \{e\} \wedge M_3 \in \{\{d\}, \{\neg d\}\}\} \cup\\
&\quad \{M_1 \cup M_2 \cup M_3 \mid M_1 \in \{\neg d\} \wedge M_2 \in \{\neg e\} \wedge M_3 \in \{\{a\}, \{\neg a\}\}\}\\
&= \{\{a, e, d\}, \{a, e, \neg d\}\} \cup \{\{\neg d, \neg e, a\}, \{\neg d, \neg e, \neg a\}\}\\
&= \{\{a, e, d\}, \{a, e, \neg d\}, \{\neg d, \neg e, a\}, \{\neg d, \neg e, \neg a\}\}
\end{aligned}$$

This leads to

$$\begin{aligned}
&\mathsf{models}(G_{51})\\
&= \{M_1 \cup M_2 \cup M_3 \mid M_1 \in \mathsf{models}(G_{51}) \wedge M_2 \in \mathsf{models}(c) \wedge M_3 \in \mathsf{models}(b)\}\\
&= \{M_1 \cup M_2 \cup M_3 \mid M_1 \in \{\{a, e, d\}, \{a, e, \neg d\}, \{\neg d, \neg e, a\}, \{\neg d, \neg e, \neg a\}\} \wedge\\
&\qquad\qquad M_2 \in \{c\} \wedge M_3 \in \{b\}\}\\
&= \{\{a, e, d, c, b\}, \{a, e, \neg d, c, b\}, \{\neg d, \neg e, a, c, b\}, \{\neg d, \neg e, \neg a, c, b\}\}
\end{aligned}$$

To obtain the set of models of G_5, the union of $\mathsf{models}(G_{51})$ *and* $\mathsf{models}(G_{52})$ *has to be constructed. We will not go into the details of the construction of* $\mathsf{models}(G_{52})$ *and directly present the result:*

$$\mathsf{models}(G_{52}) = \{\{a, e, \neg b, c, d\}, \{a, e, \neg b, c, \neg d\}, \{a, e, \neg b, \neg c, d\}, \{a, e, \neg b, \neg c, \neg d\}\}$$

Leading to

$$\mathsf{models}(G_5) = \{\{a, e, d, c, b\}, \{a, e, \neg d, c, b\}, \{\neg d, \neg e, a, c, b\}, \{\neg d, \neg e, \neg a, c, b\}, \{a, e, \neg b, c, d\}, \{a, e, \neg b, c, \neg d\}, \{a, e, \neg b, \neg c, d\}, \{a, e, \neg b, \neg c, \neg d\}\}.$$

After considering the queries presented in Section 3.1, we now see which transformations can be performed efficiently on DNNF formulae. Details on all results can be found in Darwiche (2001).

1. *Combination with the logical connectives $\neg$, $\wedge$ and $\vee$:* It is easy to see that the DNNF language is closed under disjunction. However it is not closed under negation and conjunction. It is possible to show that the DNNF language is not closed under negation by bringing forward the same arguments as in Section 3.1.2.1 where it is shown that linkless formulae are not closed under negation. A proof that the DNNF language is not closed under conjunction is presented in (Darwiche and Marquis, 2002).

2. *Conditioning and conjoining:* The DNNF language is closed under conditioning and furthermore conditioning can be computed in time linear in the size of the respective formula. From the definition of the conjoin operator and the fact that the DNNF language is closed under conditioning it follows that the DNNF language is closed under conjoining as well. Furthermore, conjoining a formula given in DNNF on a consistent conjunction of literals can be performed in time linear in the size of the formula.

3. *Minimization of a formula:* It is not possible to minimize a DNNF formula efficiently. However smooth DNNF allows an efficient minimization of formulae. Given a formula in smooth DNNF, the following minimize function can be used for minimization:

   Definition 3.1.35 (minimize Function (Darwiche, 2001))**.** *Let F, $G_i \in F^V_{prop}$, $i \in \mathbb{N}$ be propositional logic formulae in smooth DNNF. The function* $\mathsf{minimize} : F^V_{prop} \to F^V_{prop}$ *is defined as follows:*

$$\begin{aligned}
\mathsf{minimize}(F) &= F \text{ if } F \text{ is a literal, } \top \text{ or } \bot.\\
\mathsf{minimize}(\bigvee_i G_i) &= \bigvee_{\{G_i | \mathsf{mcard}(G_i) = \mathsf{mcard}(\bigvee_i G_i)\}} \mathsf{minimize}(G_i).\\
\mathsf{minimize}(\bigwedge_i G_i) &= \bigwedge_i \mathsf{minimize}(G_i).
\end{aligned}$$

According to Darwiche (2001), using the the minimize function on a formula F can be perfomed in time linear in the size of F and results in a smooth DNNF formula which corresponds to the minimization of F according to Definition 3.1.5.

4. *Projection:* The last transformation listed in Section 3.1.1.2 is projection of a formula on a set of atoms. The DNNF language is closed under projection on a set of atoms. For this, the project function presented in Definition 3.1.23 for linkless formulae can be used.

3.1.2.3 Prime Implicants and Prime Implicates

Now two different target languages for knowleimplicatesdge compilation in propositional logic are introduced: prime implicants and prime implicates. Prime implicants and prime implicates are well studied in propositional logic and there are many algorithms to compute them. See (Slagle, Chang, and Lee, 1970) and (Jackson and Pais, 1990) for examples. The next definition introduces the idea of implicants and prime implicants. In this definition, a set of literals and the conjunction of the literals of such a set are used interchangeably.

Definition 3.1.36 (Implicant/Prime Implicant/Prime Implicant Normal Form)**.** *Let KB be a propositional logic knowledge base. An* implicant *of KB is a consistent conjunction D of distinct literals such that $D \models KB$. A* prime implicant *of KB is an implicant of KB such that for every other implicant D' of KB, $D' \not\subset D$. The disjunction of all prime implicants of KB is called* prime implicant normal form *for KB.*

As stated by Cadoli and Donini (1997), implicants of a formula are very similar to models for the formula. An implicant in which every variable appears exactly once corresponds to a model with the respective assignment. Furthermore, the prime implicant normal form for a formula is a DNF of this formula.

Example 3.1.37. *Consider the propositional logic formula*

$$F = b \wedge c \wedge (a \vee d \vee (\neg c \wedge \neg a)).$$

Implicants of this formula are

$$D_1 = \{a, b, c, d\}$$
$$D_2 = \{a, b, c\}$$
$$D_3 = \{b, c, d\}$$

since for all D_i where $i \in \{1, 2, 3\}$, $D_i \models F$. D_1 is not a prime implicant of F because $D_2 \subset D_1$. In fact, both D_2 and D_3 are the only prime implicants of F resulting in the prime implicant normal form F' for F.

$$F' = (a \wedge b \wedge c) \vee (b \wedge c \wedge d)$$

Dual to implicants and prime implicants are the notions of implicates and prime implicates, which are used for knowledge compilation purposes as well.

Definition 3.1.38 (Implicate/Prime implicate/Prime implicate Normal Form)**.** *Let KB be a propositional logic knowledge base and C be a disjunction of distinct literals (a clause), not containing complementary literals. C is called an* implicat *of KB if* $KB \models C$*. C is called a* prime implicat *of KB if C is an implicate of KB and for every other implicate* C' *of KB,* $C' \not\subset C$*. The conjunction of all prime implicates of KB is called* prime implicate normal form *for KB.*

Note that the prime implicate normal form for a formula constitutes a CNF for the formula.

Computing all prime implicates of a knowledge base can be done by simply creating all possible resolvents. Each resolvent corresponds to an implicate and dismissing the non-prime ones leads to the desired result. However there are many more appropriate approaches to determine the set of prime implicates of a knowledge base. Cadoli and Donini (1997) present an overview on different approaches.

Example 3.1.39. *Consider the following CNF:*

$$F = \{\{a, b, \neg c\}, \{\neg a, b\}, \{c, d\}\}$$

In order to determine all implicates of F, all possible resolvents are constructed. For details on resolution consult (Fitting, 1996).

(1) $\{a, b, \neg c\}$
(2) $\{\neg a, b\}$
(3) $\{c, d\}$
(4) $(1 + 2) : \{b, \neg c\}$
(5) $(1 + 3) : \{a, b, d\}$
(6) $(4 + 3) : \{b, d\}$

The first three clauses correspond to the clauses given in the CNF of F. The last three clauses are obtained using resolution. All resolvents are implicates of F. However not all of them are prime implicates: (5) is not a prime implicate, since clause (6) is a proper subset of clause (5). (1) is not a prime implicate, since clause (4) is a proper subset of (1). All other clauses are prime implicates, leading to the prime implicate normal form F' *for F:*

$$F' = (\neg a \vee b) \wedge (c \vee d) \wedge (b \vee \neg c) \wedge (b \wedge d)$$

According to Chandra and Markowsky (1978), for a knowledge base containing n different variables the number of prime implicates and prime implicants is exponential in n in the worst case.

Properties of Prime Implicant Normal Form and Prime Implicate Normal Form First the different queries introduced in Section 3.1.1.1 are considered and it is investigated if they can be answered in polynomial time from formulae given in prime implicant normal form or prime implicate normal form respectively.

1. *Clausal entailment checking:*
   a) *Prime implicant normal form:* Given a knowledge base KB and a prime implicant normal form for KB. Entailment of CNF queries can be answered from the prime implicant normal form of KB in time polynomial in the size of the prime implicant normal form and the size of the query (Cadoli and Donini, 1997). More precisely, for a given knowledge base KB and a query Q in CNF, $KB \models Q$ iff every clause in Q has a nonempty intersection with every prime implicant. This can be tested efficiently.
   b) *Prime implicate normal form:* Similar to prime implicants, prime implicates allow to check entailment of CNF queries in time polynomial in the size of the prime implicate normal form and the size of the query. According to Cadoli and Donini (1997), this can be done as follows: Let KB be a knowledge base in prime implicate normal form and Q a CNF query. $KB \models Q$ iff for every non-tautological clause $C' \in Q$ there is a prime implicate C of KB with $C \models C'$ (or $C \subseteq C'$) respectively.
2. *Consistency checking:*
   a) *Prime implicant normal form:* According to Definition 3.1.36, a prime implicant is a consistent conjunction of distinct literals. This implies that every formula given in prime implicant normal form is in DNNF as well. Therefore the sat-predicate given in Definition 3.1.29 can be used for formulae in prime implicant normal form as well, yielding a linear satisfiability test for the prime implicate normal form language.
   b) *Prime implicate normal form:* As mentioned before, formulae given in prime implicate normal form allow to check clausal entailment in time polynomial to the size of the formula and the query. This implies that prime implicate normal form also offers an efficient conistency check. Consistency can be checked simply by testing if $\bot$ is entailed by the formula in prime implicate normal form.
3. *Validity checking:*
   a) *Prime implicant normal form:* From the definitions of prime implicates and implicants follows, that G is a prime implicant of formula F iff $\neg G$ is a prime implicate for $\neg F$. Therefore the negation of a formula F in prime implicants normal form is the formula $\neg F$ in prime implicate normal form. From the fact that consistency of formulae in prime implicate normal form can be tested in time polynomial to the size of the formula implies that validity of formulae in prime implicant normal form can be tested in time polynomial to the size of the formula as well.

b) *Prime implicate normal form:* It is a well-known fact that validity of a CNF formula can be checked efficiently. For this it is only necessary to check if each clause contains complementary literals and is therefore valid. This can be performed in time linear in the size of the CNF formula. Since formulae in prime implicate normal form are given in CNF, it is obvious that it is possible to check the validity of formulae in prime implicate normal in time linear in the size of the formula as well.

4. *Model enumeration:* For formulae given in prime implicate or prime implicant normal form conditioning can be performed in time polynomial to the size of the formula. See (Marquis, 2000) and Darwiche and Marquis (2002) for proofs. According to Darwiche and Marquis (2002), sublanguages of the NNF language have the following property: If the language allowing polytime conditioning and a polytime consistency check then it also allows to enumerate models in polytime. Therefore, both the prime implicate and the prime implicant normal form allow to enumerate models in time polynomial to the size of the formula.

Next we consider the different transformations given in Section 3.1.1.2.

1. *Conditioning:* As mentioned before, prime implicant and prime implicate normal form both allow polytime conditioning. See (Marquis, 2000) and (Darwiche and Marquis, 2002) for proofs.

2. *Combination with the logical connectives $\neg$, $\wedge$ and $\vee$:* As shown by Darwiche and Marquis (2002), both prime implicate and prime implicant normal form are not closed under conjunction, disjunction and negation.

3. *Projection:* When considering projection, prime implicate and prime implicant normal form do not have the same properties.

   a) *Prime implicant normal form:* It is not possible to efficiently project a formula given in prime implicant normal form on a set of variables. Darwiche and Marquis (2002) presents for an example of a formula F in prime implicant normal form for which the projection on a set of atoms has an exponentially greater number of prime implicants than F.

   b) *Prime implicate normal form:* As shown by Marquis (2000) for a formula F in prime implicate normal form and a set of variables A, it is possible to construct the projection of F on A in time polynomial to the size of F. This property is due to the fact that the set of prime implicates of the projection of F on A are exactly those prime implicates of F not containing any symbols from $\overline{A}$.

3.1.3 Relation between the Introduced Normal Forms

It is obvious that the linkless normal form and DNNF are very closely related. In fact every DNNF formula is in linkless normal form as well. Hence a formula in DNNF allows

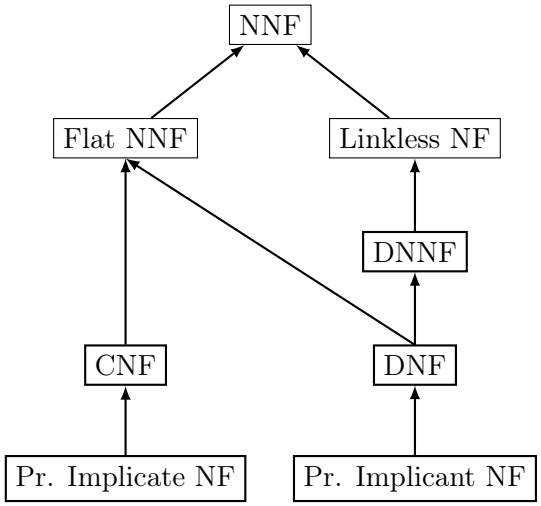


Figure 3.1: Relation between Different NNF Sublanguages. (Corresponds to Figure 4 by Darwiche and Marquis (2002) and is adapted to our purposes). An arrow from language L_1 to language L_2 means that L_1 is a proper subset of L_2.

at least the same efficient queries and transformations as a formula in linkless normal form. Since every formula in prime implicant normal form is in DNF and every DNF is in DNNF, the prime implicate normal form language is a sublanguage of the DNNF language. In order to compare the different sublanguages considered in this section, the notion of *flat NNF* is helpful. The flat NNF restricts the structure of NNF formulae: As suggested by Darwiche and Marquis (2002), a NNF formula can be interpreted as a DAG with leaf nodes labeled either with literals or with $\top$ or $\bot$. Internal nodes are labeled either with the logical connective $\wedge$ or $\vee$ which are allowed to have an arbitrary number of children. A formula is in flat NNF iff the maximal number of edges from the root to a leaf is at most two. For example both CNF and DNF formulae are in flat NNF.

Another relation that is easy to see that DNF with simple-conjunction property is a sublanguage of the DNNF language. Furthermore, the prime implicant language is a sublanguage of DNF. Figure 3.1 illustrates the relation between the different NNF sublanguages considered in this section. The figure corresponds Figure 4 by Darwiche and Marquis (2002) and is adapted to our purposes.

We end the comparison by presenting a summary of the properties of linkless normal form, DNNF, prime implicate and prime implicant normal form in Figure 3.1. For the properties in the table not explained above, see (Darwiche and Marquis, 2002) for detailed proofs.

Language	CO	VA	CE	ME	CD	FO	$\wedge$	$\vee$	$\neg$
NNF	○	○	○	○	✓	○	✓	✓	✓
Linkless NF	✓	○	✓	✓	✓	✓	○	✓	○
DNNF	✓	○	✓	✓	✓	✓	○	✓	○
Prime Implicant NF	✓	✓	✓	✓	✓	×	×	×	×
Prime Implicate NF	✓	✓	✓	✓	✓	✓	×	×	×

Content of Cells:
✓: satisfies property, ×: does not satisfy property, ○: satisfies property iff P=NP

Queries and Transformations:	
CO:	polytime consistency check
VA:	polytime validity check
CE:	polytime check of clausal entailment
ME:	polytime model enumeration
CD:	polytime conditioning
FO:	polytime forgetting
$\wedge, \vee, \neg$:	language closed under the resp. operator and resulting formula can be computed in polytime

Table 3.1: (Darwiche and Marquis, 2002) Different NNF Sublanguages and their Polynomial Time Queries and Transformations.

3.2 Knowledge Compilation in Description Logics

In this Section, methods for knowledge compilation for the description logic $\mathcal{ALC}$ are developed. In the context of description logic, one has to distinguish between knowledge compilation of concepts and knowledge compilation of TBoxes. Firstly, we focus on knowledge compilation for concepts. Section 3.2.1 introduces some normal forms, namely the so-called $\forall$-normal form, $\exists$-normal form, propagated $\exists$-normal form and the complete propagated $\exists$-normal form. These normal forms serve as a basis for a target language for $\mathcal{ALC}$ concepts called *linkless normal form* introduced in Section 3.2.2. We prove properties of this linkless normal form, such as polytime uniform interpolation. Furthermore, it is shown how the linkless normal form can serve as a target language for compiling TBoxes. The described compilation of $\mathcal{ALC}$ TBoxes into linkless normal form is implemented and Section 3.2.3.1 provides some details on the implementation together with some experimental results. Since the focus of this section is on the development of the linkless normal form and the investigation of its properties, the implementation only serves for a feasibility study and therefore no detailed evaluation is presented.

When choosing a target language for knowledge compilation purposes in description logic, several aspects have to be considered. First of all a possible blowup caused by the transformation of a concept into the normal form as well as the computational complexity of the transformation is of interest. Furthermore, like in the propositional case, tractable queries and transformations are important. Given a concept in a target language,

possible queries are consistency and validity checking as well as subsumption checking. In addition to that, transformations on concepts given in the target language can be investigated. These transformations include closure properties, conditioning, conjoining as well as projection.

Next, different normal forms for $\mathcal{ALC}$ concepts are introduced which serve as a basis for the linkless normal form which is presented in Section 3.2.2.

3.2.1 ∀- and ∃-Normal Form for $\mathcal{ALC}$

In this section, two different normal forms for $\mathcal{ALC}$ concepts will be introduced: the ∀-normal form and the ∃-normal form. These normal forms restrict the concepts which are allowed to occur in the scope of a role restriction. The ∀-normal form and propagated ∃-normal form are related to the normalization rules used by Baader, Küsters, and Molitor (1999) to compute the least common subsumer of $\mathcal{ALE}$ concept descriptions. Another related approach is the normal form used for the calculation of uniform interpolants by Wang, Wang, Topor, Pan, and Antoniou (2009). However transforming a concept into the normal form used by Wang et al. (2009) in general produces a larger blowup then the precompilation into ∃-normal form.

For the remainder of this chapter, unless stated otherwise, by the term concept, we denote $\mathcal{ALC}$ concepts given in NNF. Recall that according to Definition 2.4.4 by *literal concept*, an atomic concept or a negated atomic concept is denoted or $\bot$ or $\top$. Further by *literal* we denote a literal concept or a role restriction. By literals occurring on the topmost level of a concept C in NNF, we understand each literal occurring in C, which is not in the scope of a role restriction.

Example 3.2.1. *Consider the concept*

$$C = (A \sqcap \neg B) \sqcup \exists R.(\neg C \sqcap D)$$

Literals occurring on the topmost level of C are A, $\neg B$, $\exists R.(\neg C \sqcap D)$.

Most parts of this section have been published in (Schon, 2011) and (Schon, 2010).

3.2.1.1 ∀-Normal Form

Now we will introduce the ∀-normal form. The basic idea of this normal form is to combine all conjunctively combined universal role restrictions w.r.t. the same role. For example the concept $\forall R.A \sqcap \forall R.B$ contains the information that everything reachable via the R role has to belong to the concept A and the concept B. This information can be combined into $\forall R.(A \sqcap B)$ meaning that everything reachable via the R role has to belong to the concept $A \sqcap B$.

In order to be able to introduce the ∀-normal form, it is necessary to define the notion of conjunctively combined concepts which is similar to the notion of conjunctively combined formulae in propositional logic, given in Definition 3.1.9. An important difference is the fact that description logic concepts can contain role restrictions and it is not desired to consider for example the concepts D and E to be conjunctively combined in the concept

$\exists R.(E \sqcap \forall R.(C \sqcup D))$. The following definition of a *path* of a concept together with Definition 3.2.5 takes care of this aspect.

Definition 3.2.2 (Paths of a Concept)**.** *Let $\Sigma = (N_C, N_R, N_I)$ be a description logic signature C_1, $C_2 \in C^\Sigma$ concepts and $R \in N_R$ a role.* $\mathsf{paths} : C^\Sigma \to 2^{C^\Sigma}$ *is a function mapping a concept to a set of concepts satisfying the following conditions:*

$$\begin{aligned} \mathsf{paths}(C_1) &= \{\{C_1\}\} \text{ if } C_1 \text{ is a literal concept.} \\ \mathsf{paths}(C_1 \sqcup C_2) &= \mathsf{paths}(C_1) \cup \mathsf{paths}(C_2). \\ \mathsf{paths}(C_1 \sqcap C_2) &= \{X \cup Y \mid X \in \mathsf{paths}(C_1) \text{ and } Y \in \mathsf{paths}(C_2)\}. \end{aligned}$$

For concept C_1, $\mathsf{paths}(C_1)$ *is called the* set of paths in C_1 *and each element in* $\mathsf{paths}(C_1)$ *is called a* path in C_1.

Note that when constructing the set of paths of a given concept C, only the topmost level of C is considered. If C contains a subconcept of the form $\forall R.D$ or $\exists R.D$, paths in the subconcept D are irrelevant to the set of paths in C.

Like in the propositional case, the notion of a path and the conjunction of the elements of a path will be used interchangeably. With this understanding, it makes sense to state that a path can be satisfiable or unsatisfiable.

In propositional logic, a path is unsatisfiable iff it contains $\bot$ or complementary literals. In description logic, the situation is not that easy. Of course a path P with $\bot \in P$ or complementary literal concepts occurring as elements of P is unsatisfiable. However, in description logic it is possible for a path P to be unsatisfiable even if $\bot \notin P$ and there are no complementary literal concepts L and $\overline{L}$ with both $L \in P$ and $\overline{L} \in P$. Take the paths $\{\exists R.A, \forall R.\neg A\}$ or $\{\exists R.\bot\}$ as examples. We sum up these thoughts in the following lemma.

Lemma 3.2.3. *Let C be a concept and $P \in \mathsf{paths}(C)$. Then path P is unsatisfiable iff at least one of the following conditions is fulfilled:*

1. $\bot \in P$.
2. *there is $L \in P$ and $\overline{L} \in P$ for L a literal concepts.*
3. $\exists R.C' \in P$ *with unsatisfiable concept C'.*
4. $\{\exists R.C, \forall R.D_1, \ldots, \forall R.D_n\} \subseteq P$ *with unsatisfiable concept $C \sqcap D_1 \sqcap \ldots \sqcap D_n$.*

Proof. Lemma 3.2.3 follows directly from the way paths are constructed. □

Example 3.2.4. *Consider the concept*

$$C = \exists R.(B \sqcup E) \sqcap \forall R.\neg B \sqcap (E \sqcup D \sqcup \forall R.F)$$

which has three different paths

$$\begin{aligned} P_1 &= \{\exists R.(B \sqcup E), \forall R.\neg B, E\} \\ P_2 &= \{\exists R.(B \sqcup E), \forall R.\neg B, D\} \\ P_3 &= \{\exists R.(B \sqcup E), \forall R.\neg B, \forall R.F\} \end{aligned}$$

All paths are satisfiable.

Definition 3.2.5 (Conjunctively Combined)**.** *Let C be a concept in NNF. Concepts D and E are called* conjunctively combined *in C if there is a path $P \in \mathsf{paths}(C)$ with $D \in P$ and $E \in P$ or if C contains $QR.F$, $Q \in \{\exists, \forall\}$ and D and E are conjunctively combined in F.*

The $\forall$-normal form restricts occurrences of universal role restrictions on the topmost level of a concept. It exploits the fact that $\forall R.A \sqcap \forall R.B$ is equivalent to $\forall R.(A \sqcap B)$ and demands all conjunctively combined occurrences of universal role restrictions w.r.t. the same role to be summarized.

Definition 3.2.6 ($\forall$-Normal Form)**.** *A concept is in* $\forall$-normal form ($\forall$-NF) *if it is in NNF and the topmost level of the concept does not contain conjunctively combined concepts of the form $\forall R.B_1$ and $\forall R.B_2$.*

Example 3.2.7. *Consider concept C given in Example 3.2.4:*

$$C = \exists R.(B \sqcup E) \sqcap \forall R.\neg B \sqcap (E \sqcup D \sqcup \forall R.F).$$

C is not in $\forall$-NF, since the two universal role restrictions $\forall R.\neg B$ and $\forall R.F$ occur conjunctively combined on the topmost level of C.

Every concept can be transformed into an equivalent concept in $\forall$-NF. In order to proof this property, the notion of a disjunctive normal form for concepts is helpful:

Definition 3.2.8 (Disjunctive Normal Form)**.** *Let C be a concept in NNF. C is in disjunctive normal form (DNF) iff*

$$C = (\bigsqcup_{i=1}^{n}(\bigsqcap_{j=1}^{m} L_{i,j}))$$

where $L_{i,j}$ is a literal.

Note that this definition of DNF only affects the topmost level of a concept. For example the concept $(E \sqcap \neg B) \sqcup \exists R.(A \sqcap (D \sqcup E))$ is in DNF, even though the concept in the scope of the existential role restriction does not have a special structure. Definition 3.2.8 only claims $L_{i,j}$ to be literals and according to Definition 2.4.4 a literal can be a role restriction as well. Each concept can be transformed into DNF by transforming it into NNF and then using distributivity as well as De Morgan's laws.

Theorem 3.2.9. *For every concept there is an equivalent concept which is in $\forall$-NF.*

Proof. A concept C can be transformed into an equivalent concept C' in $\forall$-NF by first transforming it into NNF and then using the following algorithm:

1. If C doesn't contain any role restrictions, then $C' = C$.
2. If the topmost level of C contains conjunctively combined universal role restrictions $\forall R.B_1$ and $\forall R.B_2$:

a) transform C into DNF and

b) summarize all conjunctively combined universal role restrictions referring to the same role using $\forall R.B_1 \sqcap \forall R.B_2 \equiv \forall R.(B_1 \sqcap B_2)$.

It is easy to see that the application of this algorithm results in a concept in $\forall$-NF. $\square$

Example 3.2.10. *Consider concept C from Example 3.2.4:*

$$C = \exists R.(B \sqcup E) \sqcap \forall R.\neg B \sqcap (E \sqcup D \sqcup \forall R.F).$$

C can be transformed into $\forall$-NF using the algorithm introduced in the proof of Theorem 3.2.9. First C is transformed into DNF, leading to

$$\begin{aligned} C_2 = &(\exists R.(B \sqcup E) \sqcap \forall R.\neg B \sqcap E) \sqcup \\ &(\exists R.(B \sqcup E) \sqcap \forall R.\neg B \sqcap D) \sqcup \\ &(\exists R.(B \sqcup E) \sqcap \forall R.\neg B \sqcap \forall R.F)). \end{aligned}$$

Next, all conjunctively combined universal role restrictions referring to the same role are summarized. The resulting concept is:

$$\begin{aligned} C_2 = &(\exists R.(B \sqcup E) \sqcap \forall R.\neg B \sqcap E) \sqcup \\ &(\exists R.(B \sqcup E) \sqcap \forall R.\neg B \sqcap D) \sqcup \\ &(\exists R.(B \sqcup E) \sqcap \forall R.(\neg B \sqcap F)). \end{aligned}$$

However going the whole way to DNF results in a concept, which is larger than necessary. It is possible to create a more succinct version of the $\forall$-NF of a concept by expanding the concept only as far as necessary into DNF. Where necessary means, that C is gradually transformed using distributivity as well as De Morgan's laws only until the $\forall$-NF is reached. For the concept considered above, it is possible to compute a more succinct $\forall$-NF which is:

$$C_3 = \exists R.(B \sqcup E) \sqcap \big((\forall R.\neg B \sqcap (E \sqcup D)) \sqcup \forall R.(\neg B \sqcap F)\big). \tag{3.1}$$

One way to transform a concept into $\forall$-NF is using the idea of path dissolution from propositional logic (Murray and Rosenthal, 1993). Usually path dissolution is used to remove paths containing a link from a propositional logic formula. Path dissolution can be also used to remove conjunctively combined universal role restrictions $\forall R.D$ and $\forall R.E$ from a concept C. For this, we use the bijective function prop presented in Definition 2.4.36 that maps

- each atomic concept A to a propositional logic variable a,
- $\sqcap$ to $\wedge$,
- $\sqcup$ to $\vee$,
- $\top$ to *true*,

- $\bot$ to *false*,
- $\neg$ to $\neg$ and
- $QR.C$ to a propositional logic variable Qrc with $Q \in \{\exists, \forall\}$.

In order to remove one occurrence of conjunctively combined universal role restrictions $\forall R.D$ and $\forall R.E$ from a concept C, firstly we determine the smallest subconcept $G \sqcap H$ of C modulo commutativity, which contains the conjunctive combination of $\forall R.D$ and $\forall R.E$. Modulo commutativity of $\sqcap$ means, that there is a subconcept $D_1 \sqcap \ldots \sqcap D_n$ of C with $n \geq 2$ and there are distinct D_i, D_j in $\{D_1, \ldots, D_n\}$ with $G = D_i$ and $H = D_j$. W.l.o.g. we assume that $\forall R.D$ occurs in G and $\forall R.E$ occurs in H. Next the bijective function prop is used to construct the propositional logic formula

$$\mathsf{prop}(G \sqcap H) = G' \wedge H'.$$

Then $PE(\forall rd, G')$, $PC(\forall rd, G')$, $PE(\forall re, H')$ and $PC(\forall re, H')$ as given in Definition 3.1.13 is constructed. It follows from Lemma 3.1.14 that

$$\begin{aligned} G' \equiv & PE(\forall rd, G') \vee PC(\forall rd, G') \\ H' \equiv & PE(\forall re, H') \vee PC(\forall re, H'). \end{aligned}$$

Therefore $G' \wedge H'$ is equivalent to

$$\begin{aligned} & \big(PC(\forall rd, G') \wedge PC(\forall re, H')\big) \vee \\ & \big(PC(\forall rd, G') \wedge PE(\forall re, H')\big) \vee \\ & \big(PE(\forall rd, G') \wedge PC(\forall re, H')\big) \vee \\ & \big(PE(\forall rd, G') \wedge PE(\forall re, H')\big). \end{aligned} \tag{3.2}$$

Note that only the last disjunct of formula (3.2) contains conjunctively combined occurrences of $\forall rd$ and $\forall re$ and further every path in the last disjunct contains both $\forall rd$ and $\forall re$. In the next step, the bijective function prop is used to map formula (3.2) back to a concept called N. Since every path in

$$\big(PE(\forall rd, G') \wedge PE(\forall re, H')\big)$$

contains both $\forall rd$ and $\forall re$, they can be combined to $\forall R.(D \sqcap E)$ in the concept N. Next we substitute the result of this for $G \sqcap H$ in the original concept C. After this step the number of conjunctively combined concepts of the form $\forall R.C_1$ and $\forall R.C_2$ in C decreased by one.

In this way, all conjunctively combined concepts of the form $\forall R.C_1$ and $\forall R.C_2$ in C can be removed step by step, leading to a concept in $\forall$-NF. Note that the result of this transformation does not necessarily have to be in DNF. Only in the worst case, the result is in DNF which means an exponential blowup occurred. Murray and Rosenthal (1993) present many optimizations for path dissolution which help to keep the result of dissolution as succinct as possible.

3.2.1.2 ∃-Normal Form

Next we introduce the ∃-normal form which imposes restrictions on the occurrences of existentially quantified role restrictions on the topmost level of a concept.

Definition 3.2.11 (∃-Normal Form)**.** *A concept C is in* ∃-normal form (∃-NF) *if C is in ∀-NF and further for each occurrence of a subconcept $\exists R.B$ on the topmost level of C the following conditions hold:*

- *the occurrence of $\exists R.B$ is conjunctively combined with at most one concept of the form $\forall R.D$,*
- *if the occurrence of $\exists R.B$ is conjunctively combined with a concept of the form $\forall R.D$, then all paths in C containing this occurrence of $\exists R.B$ also contain $\forall R.D$.*

The definition of the ∃-NF slightly deviates from the definitions presented in (Schon, 2011) and (Schon, 2010). Definition 3.2.11 ensures that if an occurrence of $\exists R.B$ is contained in a path which also contains $\forall R.E$, then all paths containing this occurrence of $\exists R.B$ also contain $\forall R.E$. Ensuring for example that concept $\exists R.C \sqcap (A \sqcup \forall R.B)$ is not in ∃-NF, since $\exists R.C$ is conjunctively combined with $\forall R.B$ but is furthermore contained in a path which does not contain $\forall R.B$.

Note that Definition 3.2.11 restricts *occurrences* of concepts of the form $\exists R.B$ on the topmost level of C. This means that for example the concept

$$C = (\exists R.B \sqcap \forall R.(E \sqcup F)) \sqcup (\exists R.B \sqcap \forall R.\neg E)$$

is in ∃-NF because the claimed condition holds for both occurrences of $\exists R.B$ in C, even though $\exists R.B$ occurs conjunctively combined with $\forall R.(E \sqcup F)$ and with $\forall R.\neg E$.

Theorem 3.2.12. *For every concept there is an equivalent concept which is in ∃-NF.*

Proof. The proof of Theorem 3.2.12 can be done analog to the proof of Theorem 3.2.9. □

The ∃-NF of a concept can be computed by an algorithm which is similar to the one given in the proof of Theorem 3.2.9. As in the case of the ∀-NF this can lead to an ∃-NF which is larger than necessary. However by partially expanding the concept only as far as necessary it is possible to produce a more succinct ∃-NF. Like described for the ∀-NF, it is possible to use path dissolution to compile a concept given in ∀-NF into ∃-NF.

Example 3.2.13. *Consider C_3 given in (3.1) in Example 3.2.10 as:*

$$C_3 = \exists R.(B \sqcup E) \sqcap \big((\forall R.\neg B \sqcap (E \sqcup D)) \sqcup \forall R.(\neg B \sqcap F)\big).$$

The following concept C_4 is equivalent to C_3 and is in ∃-NF:

$$C_4 = \big(\exists R.(B \sqcup E) \sqcap \forall R.\neg B \sqcap (E \sqcup D)\big) \sqcup \big(\exists R.(B \sqcup E) \sqcap \forall R.(\neg B \sqcap F)\big). \tag{3.3}$$

With the help of the following lemma, existential and universal role restrictions occurring conjunctively combined in a concept can be summarized.

Lemma 3.2.14. *Let $\Sigma = (N_C, N_R, N_i)$ be a description logic signature, C, $D \in N_C$ and $R \in N_R$. Then*

$$\exists R.C \sqcap \forall R.D \equiv \exists R.(C \sqcap D) \sqcap \forall R.D$$

Proof. Lemma 3.2.14 follows immediatly from the semantics of $\mathcal{ALC}$ given in Definition 2.4.8. □

Definition 3.2.15 (Propagated ∃-Normal Form/Complete Propagated ∃-Normal Form)**.** *Let C be a concept in ∃-NF. C is in* propagated ∃*-normal form (propagated ∃-NF) if for all concepts of the form $\exists R.C_1$ and $\forall R.C_2$ occurring conjunctively combined in C, C_1 is equivalent to $C_1 \sqcap C_2$. Further C is in* complete propagated ∃*-NF if C is in propagated ∃-NF and for all $QR.B$ occurring in C with $Q \in \{\exists, \forall\}$, B is in complete propagated ∃-NF as well.*

Every concept in ∃-NF can be transformed into an equivalent concept in propagated ∃-NF with the help of Lemma 3.2.14.

Example 3.2.16. *Reconsider concept C_4 given in (3.3) in Example 3.2.13.*

$$C_4 = \big(\exists R.(B \sqcup E) \sqcap \forall R.\neg B \sqcap (E \sqcup D)\big) \sqcup \big(\exists R.(B \sqcup E) \sqcap \forall R.(\neg B \sqcap F)\big).$$

C_4 is equivalent to concept C_5 which is in propagated ∃-NF:

$$C_5 = \big(\exists R.((B \sqcup E) \sqcap \neg B) \sqcap \forall R.\neg B \sqcap (E \sqcup D)\big) \sqcup \big(\exists R.((B \sqcup E) \sqcap \neg B \sqcap F) \sqcap \forall R.(\neg B \sqcap F)\big)$$

From a knowledge compilation point of view, the complete propagated ∃-NF does not have interesting properties. However it can be used as a basis to transfer the idea of some known target languages for knowledge compilation in propositional logic to description logic concepts. In the following, the idea of the linkless normal form as introduced by Murray and Rosenthal (1993) is used to demonstrate this aspect. The basic idea is to first transform a concept into its complete propagated ∃-NF and afterwards use the bijective function prop and path dissolution to remove all links from the topmost level of the concept and all subconcepts D occurring in the form of $\exists R.D$ or $\forall R.D$.

3.2.2 Linkless Normal Form for $\mathcal{ALC}$ Concepts

The complete propagated ∃-NF introduced in Section 3.2.1.2 will now serve as a basis to transfer the notion of linkless formulae known from propositional logic to $\mathcal{ALC}$ concepts. In this section, a normal form called *linkless normal form* is introduced. To obtain this normal form, first the topmost level of a concept given in complete propagated ∃-NF is compiled by removing so called *links*. In the next step, we will recursively perform the precompilation on subconcepts occurring in the scope of a role restriction.

Most parts of this section have been published in (Schon, 2011), (Schon, 2010), (Furbach and Obermaier, 2008) and (Furbach et al., 2009).

We start by giving a definition of a link in a concept. Like in propositional logic, a link is a contradictory part of a concept, which can be removed from the concept preserving equivalence.

Definition 3.2.17. *Let C be a concept. A* link *in C is a set of two complementary literal concepts occurring in a path of C. C is called* top level linkless *if C is in NNF and there is no path in C which contains a link.*

In propositional logic, a linkless formula F is satisfiable if F is simplified according to the last two equivalences in Theorem 2.1.10 as far as possible and $F \neq \bot$. In description logic, a top level linkless concept is not necessarily satisfiable. Take the top level linkless concept $\exists R.(\neg B \sqcap B) \sqcap \forall R.B$ as an example. We will address this problem later. Firstly, a method to remove links from a concept will be introduced.

As in propositional logic, path dissolution (Murray and Rosenthal, 1993) can be used to remove links from a description logic concept C. For this C has to be mapped to a propositional logic formula using the bijective function prop introduced in Definition 2.4.36. To this propositional logic formula $\mathsf{prop}(C)$, the $\mathsf{fulldissolvent}$ function from Definition 3.1.20 is applied leading to the linkless propositional logic formula $\mathsf{fulldissolvent}(\mathsf{prop}(C))$. In the next step, the inverse of prop is used to map the result back to a description logic concept. Leading to the top level linkless concept $\mathsf{prop}^{-1}(\mathsf{fulldissolvent}(\mathsf{prop}(C)))$. To ease readability, we define the function $\mathsf{fulldissolventDL}$ performing the above mentioned steps to gain a top level linkless concept.

Definition 3.2.18 ($\mathsf{fulldissolventDL}$)**.** *Let C be a concept in propagated $\exists$-NF. Then* $\mathsf{fulldissolventDL}(C)$ *is the top level linkless concept obtained by calculating* $\mathsf{prop}^{-1}(\mathsf{fulldissolvent}(\mathsf{prop}(C)$ *using the* $\mathsf{fulldissolvent}$ *function given in Definition 3.1.20.*

Theorem 3.2.19. *Let C be a concept. Then* $\mathsf{fulldissolventDL}(C) \equiv C$.

Proof. Theorem 3.2.19 follows from the fact that path dissolution preserves equivalence in the propositional case together with the fact that the prop function is bijective. □

In both propositional logic and description logic a path P is unsatisfiable if the conjunction of its elements is unsatisfiable. A link in a path clearly indicates an unsatisfiability. However, in contrast to propositional logic, a path of a concept not containing any links and not containing $\bot$ is not necessarily satisfiable. For a concept given in propagated $\exists$-NF, a path P is unsatisfiable iff P contains a link or $\bot$ or P contains $\exists R.C$ where C is an unsatisfiable concept. Therefore, in order to get a normal form for description logic concepts with the nice properties of linkless normal form for propositional logic formulae, it is necessary to take concepts occurring in the scope of a role restriction into account as well. Informally speaking, this can be achieved by removing links not only from the topmost level of the concept but from *all levels* of the concept.

To develop a linkless normal form for concepts, the concept is assumed to be in propagated $\exists$-NF. This is an important requirement since it ensures that a path is unsatisfiable iff it contains a link or $\bot$ or it contains $\exists R.C$ where C is unsatisfiable. For a concept not given in propagated $\exists$-NF, this is not necessarily true. We illustrate this with the following example:

$$\top \sqcap D \equiv D \qquad \top \sqcup D \equiv \top \qquad \exists R.\bot \equiv \bot$$
$$\bot \sqcap D \equiv \bot \qquad \bot \sqcup D \equiv D \qquad \forall R.\top \equiv \top$$

Figure 3.2: Equivalences Used for Simplification, where D is a Concept and R a Role.

Example 3.2.20. *Consider the following concept C which is not in propagated $\exists$-NF:*

$$C = \exists R.(\neg B \sqcap \neg E) \sqcap \forall R.E$$

C is unsatisfiable and has the single path

$$P = \{\exists R.(\neg B \sqcap \neg E),\, \forall R.E\}$$

P contains the existential role restriction $\exists R.(\neg B \sqcap \neg E)$. However the concept $\neg B \sqcap \neg E$ occurring in the scope of the role restriction is not unsatisfiable. Transforming C into complete propagated $\exists$-NF leads to

$$C' = \exists R.(\neg B \sqcap \neg E \sqcap E) \sqcap \forall R.E$$

which has the path

$$P' = \{\exists R.(\neg B \sqcap \neg E \sqcap E),\, \forall R.E\}$$

containing the existential role restriction $\exists R.(\neg B \sqcap \neg E \sqcap E)$. The concept $\neg B \sqcap \neg E \sqcap E$ clearly is unsatisfiable.

With the help of these preliminary considerations, we are now able to present the definition of linkless normal form for description logic concepts.

Definition 3.2.21 (Linkless Normal Form)**.** *Let C be a concept in complete propagated $\exists$-NF. C is in* linkless normal form (linkless NF) *if C is top level linkless and for all $QR.D$, where $Q \in \{\exists, \forall\}$, occurring in C, D is in linkless NF and further C is simplified according to the equivalences given in Figure 3.2.*

For a concept C, we call a linkless concept D which is equivalent to C, a *linkless version of C*. A concept, which is given in linkless NF is also called linkless. Note that a linkless concept C can only be unsatisfiable if $C = \bot$.

The way linkless NF is defined ensures that not only the topmost level of a linkless concept does not contain any links but all concepts occurring in the scope of role restrictions are linkless as well. This guarantees that, whenever during reasoning a subconcept of the form $QR.D$, where $Q \in \{\exists\forall\}$, has to be considered, D is linkless and therefore has nice computational properties. As mentioned before, a path of a concept C in propagated $\exists$-NF is unsatisfiable iff it contains a link or $\bot$ or a role restriction $\exists R.D$ where D is unsatisfiable. This implies that, when checking satisfiability of C it makes sense that it is helpful for D to be in linkless normal form.

According to Definition 3.2.21, all subconcepts occurring in the scope of an universal role restriction have to be linkless as well. This requirement is a little bit less intuitive,

but its necessity gets more comprehensible as soon as subsumption tests are involved. Given a linkless concept C and another concept D, checking if $C \sqsubseteq D$ can introduce new existential role restrictions, which need to be combined with universal role restrictions occurring in C. Therefore, it is advantageous to have linkless versions of concepts occurring in the scope of universal role restrictions.

Example 3.2.22. *To illustrate the linkless normal form, we now present three different concepts and examine if they are in linkless normal form:*

- $C_1 = \Big(\exists R.\big((A \sqcap C) \sqcup \neg B\big) \sqcup E\Big) \sqcap \forall R.\neg C$

  C_1 is top level linkless. However it is not in propagated $\exists$-NF and therefore C_1 is not linkless.

- $C_2 = \forall R.D \sqcap \exists R.\big((E \sqcup (B \sqcap \neg D)) \sqcap D\big)$

  C_2 is top level linkless and in propagated $\exists$-NF. However the concept $(E \sqcup (B \sqcap \neg D)) \sqcap D$ occurring in the scope of the existential role restriction is not top level linkless and therefore C_2 is not linkless.

- $C_3 = \big(A \sqcup (\neg B \sqcap C)\big) \sqcap \exists R.\big((A \sqcup C) \sqcap \neg D\big) \sqcap \forall R.\neg D$

  C_3 is linkless, since it is top level linkless, in propagated $\exists$-NF and all concepts occurring in the scope of role restrictions are linkless.

Theorem 3.2.23. *Every concept can be transformed into an equivalent linkless concept.*

Proof. Let C be a concept and D be the concept resulting from applying the following steps to C.

1. Transform C into complete propagated $\exists$-NF.
2. Replace C by $\mathsf{fulldissolventDL}(C)$.
3. For all role restrictions $QR.B$ on the topmost level of C, where $Q \in \{\exists, \forall\}$, replace B by the linkless version of of B.
4. Simplify C according to the equivalences given in Figure 3.2.

Obviously concept D is linkless. Furthermore, $C \equiv D$ follows from Theorem 3.2.19 together with the fact that every concept can be transformed into an equivalent concept in complete propagated $\exists$-NF. □

The proof of Theorem 3.2.23 gives an algorithm to compile a concept into an equivalent linkless concept. Performing the simplifications of Figure 3.2 during this algorithm ensures that a linkless concept can only be unsatisfiable if it is $\bot$. Since dissolution only removes unsatisfiable paths from the concept and does not introduce new paths, it is guaranteed that all levels of a linkless concept are in propagated $\exists$-NF.

Example 3.2.24. *Consider concept C_5 from Example 3.2.16, which is given in propagated ∃-NF:*

$$C_5 = \big(\exists R.((B \sqcup E) \sqcap \neg B) \sqcap \forall R.\neg B \sqcap (E \sqcup D)\big) \sqcup \big(\exists R.((B \sqcup E) \sqcap \neg B \sqcap F) \sqcap \forall R.(\neg B \sqcap F)\big).$$

The linkless normal form for C_5 is

$$C_6 = \big(\exists R.(E \sqcap \neg B) \sqcap \forall R.\neg B \sqcap (E \sqcup D)\big) \sqcup \big(\exists R.(E \sqcap \neg B \sqcap F) \sqcap \forall R.(\neg B \sqcap F)\big).$$

When considering possible target languages for knowledge compilation purposes, a possible blowup caused by the transformation is of special interest. We will now show that transforming a concept into linkless normal form causes an exponential blowup in the worst case. To see why this is the case, it is helpful to first consider flattening of a concept. This is a transformation normally used for TBoxes (Rudolph, Krötzsch, and Hitzler, 2008) which removes nested role restrictions. It can be used to get rid of nested role restrictions in concepts as well. We adapt the definition of flattening for TBoxes introduced by Rudolph et al. (2008) to flattening of concepts in the following definition.

Definition 3.2.25 (Flattening of a Concept)**.** *Let Σ be a description logic signature and $C \in C^\Sigma$ be a concept. Function $\mathsf{flat} : C^\Sigma \to C^\Sigma$ is a function removing nested role restrictions from concepts. $\mathsf{flat}(C)$ is the result of repeating the following steps until there are no more nested role restrictions in C:*

1. *Select an outermost occurrence of a role restriction $QR.D$, $Q \in \{\exists, \forall\}$ and D not a literal concept.*
2. *Replace each occurrence of $QR.D$ in C by $QR.F$ with F a fresh atomic concept.*
3. *Conjunctively add $\neg F \sqcup D$ to C.*

$\mathsf{flat}(C)$ is called the flattening of C.

Example 3.2.26. *Consider the following concept C containing a nested role restriction:*

$$C = \exists R.(\forall R.\neg E) \sqcap D$$

Flattening C leads to

$$\mathsf{flat}(C) = \exists R.F \sqcap D \sqcap (\neg F \sqcup \forall R.\neg E)$$

which does not contain any nested role restrictions.

By a simple induction on the number of role restrictions occurring in C it can be shown that flattening a concept C containing k role restrictions increases the size of C with $5k$ times in the worst case. According to Rudolph et al. (2008), the result of flattening is equisatisfiable to the original concept.

When considering the blowup caused by transforming a concept C into linkless normal form, w.l.o.g. we assume C to be flattened. Since flattening only causes a linear blowup, this assumption is not harmful. Linkless normal form requires complete propagated $\exists$-NF. Therefore a blowup caused by the transformation into propagated $\exists$-NF is of interest as well. For the transformation into complete propagated $\exists$-NF and for the removal of links, path dissolution comes to use. Path dissolution created a DNF in the worst case. In this case, an exponential blowup is caused. Therefore transforming a concept C into linkless normal form in the worst case causes an exponential blowup as well.

Next we consider properties of linkless concepts namely some of the transformations and queries introduced in Section 3.1 for propositional logic.

3.2.2.1 Properties of Linkless Concepts

In this section, different properties of linkless concepts are investigated in order to clarify the benefits of the linkless NF for concepts. For this, the different queries and transformation introduced in Section 3.1 for propositional logic are considered.

Consistency Checking The first tractable query that we consider is consistency checking. Since the simplifications of Figure 3.2 are preformed during the transformation into linkless normal form, a linkless concept can only be unsatisfiable if it is $\bot$.

Theorem 3.2.27. *Let C be a linkless concept. Then C is unsatisfiable iff $C = \bot$.*

Proof. Let C be a linkless concept. We have to show that 1. if $C = \bot$, C is unsatisfiable and 2. if C is unsatisfiable, $C = \bot$. 1.) follows immediately. We prove 2.) by induction on the nesting depth of role restrictions $\mathsf{depth}(C)$ in C:

Induction basis: $\mathsf{depth}(C) = 0$, meaning that C does not contain any role restrictions. Since C is linkless, C can only be unsatisfiable if $\bot$ is an element of every path. Furthermore all linkless concepts are simplified according to the equivalences in Figure 3.2. Therefore $C = \bot$ follows.

Induction hypothesis: For linkless concepts C with $\mathsf{depth}(C) \leq n$ holds: If C is unsatisfiable, then $C = \bot$.

Induction step: Let C be unsatisfiable, linkless and $\mathsf{depth}(C) = n + 1$. We have to show $C = \bot$. Therefore C is assumed to be unsatisfiable. A concept is unsatisfiable iff all its paths are unsatisfiable. Lemma 3.2.3 lists four conditions of which at least one must be fulfilled for a path to be unsatisfiable. We will show that for all paths P in C only the first reason, namely that P contains $\bot$ is possible in our case. For this let P be an arbitrary path in C. Since C is unsatisfiable, P has to be unsatisfiable as well. Therefore, P has to fulfill at least one of the conditions of Lemma 3.2.3:

Condition 4.): $\{\exists R.C, \forall R.D_1, \ldots, \forall R.D_n\} \subseteq P$ with unsatisfiable concept $C \sqcap D_1 \sqcap \ldots \sqcap D_n$. However since C is in propagated $\exists$-NF, C has to be equivalent to

$C \sqcap D_1 \sqcap \ldots \sqcap D_n$ meaning that C itself is unsatisfiable. This corresponds to condition 3.) of Lemma 3.2.3.

Condition 3.): P contains $\exists R.C_1$ with unsatisfiable C_1. Since $\mathsf{depth}(C_1) \leq n$, the induction hypothesis can be applied to C_1 leading to $C_1 = \bot$ meaning that P contains $\exists R.\bot$. This is a contradiction to the fact that C is linkless and therefore has to be simplified according to the equivalences in Figure 3.2. It follows that P cannot fulfill condition 3.).

Condition 2.): P contains complementary literals. However this is not possible since C is linkless and therefore does not contain complementary literal concepts in a path. Hence P cannot fulfill this condition.

Condition 1.): Since P is unsatisfiable and does not fulfill condition 2.) - 4.), P has to fulfill condition 1.) meaning that P contains $\bot$. From the fact that C is in linkless normal form and therefore is simplified according to the equivalences in Figure 3.2 follows $C = \bot$.

□

The next corollary is an immediate consequence of Theorem 3.2.27.

Corollary 3.2.28. *Let C be a linkless concept. Then consistency of C can be tested in constant time.*

Closure Properties One transformation interesting for target languages for knowledge compilation listed in Section 3.2 are closure properties. From the structure of linkless concepts follows the next theorem:

Theorem 3.2.29. *Let C_1 and C_2 be linkless concepts. Then $C_1 \sqcup C_2$, $\forall R.C_1$ and $\exists R.C_1$ are linkless as well.*

It is easy to see that linkless concepts are not closed under negation, since the negation of a linkless concept generally is not in NNF and applying De Morgan's laws to create NNF can introduce new links. Further linkless concepts are not closed under conjunction. Take the linkless concepts A and $\neg A$ as an example: $A \sqcap \neg A$ is not linkless.

Subsumption Checking Next, we consider subsumption checking. In the context of knowledge compilation this corresponds to the following task: Given a linkless concept C and a concept E, check if $C \sqsubseteq E$. In general, a subsumption $C \sqsubseteq E$ holds iff $C \sqcap \neg E$ is unsatisfiable.

To simplify notation only subsumptions $C \sqsubseteq \neg D$ are considered, which hold iff $C \sqcap D$ is unsatisfiable. Given a linkless concept C, a subsumption $C \sqsubseteq \neg D$ can be answered in time linear in $\mathsf{size}(C) \cdot \mathsf{size}(D)$ if D has a certain structure, namely is a satisfiable $\mathcal{ALE}$ concept in complete propagated $\exists$-NF for which all subconcepts occurring in the scope of role restrictions are satisfiable. The next definition specifies these concepts D as so called *q-concepts*.

Definition 3.2.30 (q-Concept)**.** *Let D be an $\mathcal{ALE}$ concept. D is called a* q-concept *if D is satisfiable, in complete propagated $\exists$-NF and for all $QR.E$ occurring in D, E is satisfiable.*

When considering a subsumption check $C \sqsubseteq \neg D$, we assume D to be much smaller than concept C. Therefore, a possible exponential blowup produced by the transformation of D into propagated $\exists$-NF is not too harmful.

As mentioned above, in order to check a subsumption query $C \sqsubseteq \neg D$ the consistency of $C \sqcap D$ has to be checked. Concept C is assumed to be a linkless concept. However linkless concepts are not closed under conjunction. Hence the concept $C \sqcap D$ doesn't need to be linkless. This is why an operator is needed, which allows to conjunctively combine the linkless concept C with the q-concept D resulting in a linkless concept. The operator used here is an enhancement of the conditioning operator introduced by Darwiche (2001) for propositional logic formulae. Intuitively, conditioning a linkless concept C by a q-concept D means, that we assume D to be true and simplify C according to this assumption. In the following we use a q-concept D and the set of its conjuncts interchangeably.

Definition 3.2.31 (Conditioning)**.** *Let C be a linkless concept and D be a q-concept. Then C conditioned with D, denoted by $C \mid D$, is defined as:*

1. *If C is a literal concept or $C \in \{\bot, \top\}$:*
$$C \mid D = \begin{cases} \top & \text{if } C \in D. \\ \bot & \text{if } \overline{C} \in D. \\ C & \text{otherwise.} \end{cases}$$

2. *If C has the form $C_1 \sqcap C_2$:*
$$C \mid D = \begin{cases} \bot & \text{if } C_1 \mid D = \bot \text{ or } C_2 \mid D = \bot. \\ C_i \mid D & \text{if } C_j \mid D = \top, \text{ where } i, j \in \{1, 2\} \text{ and } i \neq j. \\ (C_1 \mid D) \sqcap (C_2 \mid D) & \text{otherwise.} \end{cases}$$

3. *If C has the form $C_1 \sqcup C_2$:*
$$C \mid D = \begin{cases} \top & \text{if } C_1 \mid D = \top \text{ or } C_2 \mid D = \top. \\ (C_i \mid D) & \text{if } C_j \mid D = \bot, \text{ where } i, j \in \{1, 2\} \text{ and } i \neq j). \\ (C_1 \mid D) \sqcup (C_2 \mid D) & \text{otherwise.} \end{cases}$$

4. *If C has the form $\forall R.E$:*
$$C \mid D = \begin{cases} \bot & \text{if } \exists R.B' \in D \text{ with } E \mid B' = \bot. \\ \forall R.(E \mid B) & \text{if } \forall R.B \in D \text{ and there is no } \exists R.B' \in D \text{ with } E \mid B' = \bot. \\ \forall R.E & \text{otherwise.} \end{cases}$$

5. *If C has the form $\exists R.E$:*
$$C \mid D = \begin{cases} \bot & \text{if } \forall R.B \in D \text{ and } E \mid B = \bot. \\ \exists R.(E \mid B) & \text{if } \forall R.B \in D \text{ and } E \mid B \neq \bot. \\ \exists R.E & \text{otherwise.} \end{cases}$$

Note that $C \mid D$ is linkless. Conditioning a concept C by a q-concept D can be performed in time linear in $\mathsf{size}(C) \cdot \mathsf{size}(D)$.

Example 3.2.32. *To give an intuition of the conditioning operator, we consider different linkless concepts which are conditioned by different q-concepts:*

$$(E \sqcap (\neg B \sqcup \forall R.D)) | \neg E = \neg B \sqcup \forall R.D$$

$$\forall R.(\neg B \sqcup F) \mid \forall R.B = \forall R.F$$
$$\forall R.(\neg B \sqcap F) \mid \exists R.C = \forall R.(\neg B \sqcap F)$$

$$\exists R.(E \sqcap \neg B) \mid \forall R.B = \bot$$
$$\exists R.(E \sqcap \neg B) \mid \forall R.E = \exists R.\neg B$$

$$\exists R.(E \sqcap B) \sqcap \forall R.B \mid \forall R.\neg B = \bot$$

Conditioning can be used to answer queries of the form $C \sqsubseteq \neg D$ for linkless concepts C and q-concepts D. To prove this, we introduce some lemmas, namely Lemma 3.2.33, Lemma 3.2.34, Lemma 3.2.35 and Lemma 3.2.36. Lemma 3.2.33 states a connection between conditioning and conjunction and Lemma 3.2.34 can be used to simplify satisfiability checking of conjunctions of linkless formulae and q-concepts. Lemma 3.2.35 states that the result of conditioning is in propagated $\exists$-NF. Lemma 3.2.36 establishes a connection between conditioning and conjunction and is used to prove the fact that queries of the form $C \sqsubseteq \neg D$ for linkless concepts C and q-concepts D can be answered using conditioning which is stated in Theorem 3.2.37.

Lemma 3.2.33. *Let C be a linkless concept and D a q-concept. Then*

$$(C \mid D) \sqcap D \equiv C \sqcap D.$$

Proof. Let C be a linkless concept and D a q-concept. We have to show that $(C \mid D) \sqcap D \equiv C \sqcap D$. We prove this by induction on the structure of concept C:

Induction basis: C is a literal concept or $C = \bot$ or $C = \top$:

a.) $C \in D$:
Then $(C \mid D) \sqcap D \overset{\text{Def. 3.2.31}}{=} \top \sqcap D \equiv D$. Further $C \sqcap D \equiv D$ because $C \in D$.

b.) $\overline{C} \in D$:
Then $(C \mid D) \sqcap D \overset{\text{Def. 3.2.31}}{=} \bot \sqcap D \equiv \bot$. Further $C \sqcap D \equiv \bot$ because $\overline{C} \in D$.

c.) $\overline{C} \notin D$ and $C \notin D$:
Then $(C \mid D) \sqcap D \overset{\text{Def. 3.2.31}}{=} C \sqcap D$.

Induction hypothesis: For linkless concepts C_1, C_2, for all q-concepts D: $(C_i \mid D) \sqcap D \equiv C_i \sqcap D$ $(i \in \{1, 2\})$. In the remainder of this proof $\overset{\text{IH}}{\equiv}$ indicates that this equivalence follows from the induction hypothesis.

Induction step: We now have to show, that the assertion applies to the linkless concepts $C_1 \sqcap C_2$, $C_1 \sqcup C_2$, $\forall R.C_1$ and $\exists R.C_1$. For each of these cases, we examine the cases for the structure of C given in Definition 3.2.31.

1.) $C = C_1 \sqcap C_2$

a.) $C_1 \mid D = \bot$ or $C_2 \mid D = \bot$. W.l.o.g. let $C_1 \mid D = \bot$. Then

$$C_1 \mid D \sqcap D = \bot \sqcap D \equiv \bot. \tag{3.4}$$

Further $(C_1 \mid D) \sqcap D \overset{\text{IH}}{\equiv} C_1 \sqcap D$ which together with (3.4) implies

$$C_1 \sqcap D \equiv \bot. \tag{3.5}$$

Then

$$\begin{aligned}
(C \mid D) \sqcap D &\overset{\text{Def. 3.2.31}}{=} \bot \sqcap D \\
&\equiv \bot \\
&\equiv \bot \sqcap C_2 \\
&\overset{(3.5)}{\equiv} (C_1 \sqcap D) \sqcap C_2 \\
&\equiv (C_1 \sqcap C_2) \sqcap D \\
&\equiv C \sqcap D
\end{aligned}$$

b.) $C_1 \mid D = \top$ or $C_2 \mid D = \top$. W.l.o.g. let $C_1 \mid D = \top$. Then

$$\begin{aligned}
(C \mid D) \sqcap D &\overset{\text{Def. 3.2.31}}{=} C_2 \mid D \sqcap D \\
&\overset{\text{IH}}{\equiv} C_2 \sqcap D \\
&\equiv D \sqcap (C_2 \sqcap D) \\
&\equiv (\top \sqcap D) \sqcap (C_2 \sqcap D) \\
&\equiv (C_1 \mid D \sqcap D) \sqcap (C_2 \sqcap D) \\
&\overset{\text{IH}}{\equiv} (C_1 \sqcap D) \sqcap (C_2 \sqcap D) \\
&\equiv (C_1 \sqcap C_2) \sqcap D \\
&\equiv C \sqcap D
\end{aligned}$$

c.) Otherwise:

$$\begin{aligned}
(C \mid D) \sqcap D &\overset{\text{Def. 3.2.31}}{=} (C_1 \mid D) \sqcap (C_2 \mid D) \sqcap D \\
&\equiv ((C_1 \mid D) \sqcap D) \sqcap ((C_2 \mid D) \sqcap D) \\
&\overset{\text{IH}}{\equiv} (C_1 \sqcap D) \sqcap (C_2 \sqcap D) \\
&\equiv (C_1 \sqcap C_2) \sqcap D \\
&\equiv C \sqcap D
\end{aligned}$$

2.) $C = C_1 \sqcup C_2$

a.) $C_1 \mid D = \top$ or $C_2 \mid D = \top$. W.l.o.g. let $C_1 \mid D = \top$. Then

$$
\begin{array}{lll}
(C \mid D) \sqcap D & \overset{\text{Def. 3.2.31}}{=} & \top \sqcap D \\
& \equiv & D \\
& \equiv & D \sqcup (C_2 \sqcap D) \\
& \equiv & (\top \sqcap D) \sqcup (C_2 \sqcap D) \\
& \equiv & ((C_1 \mid D) \sqcap D) \sqcup (C_2 \sqcap D) \\
& \overset{\text{IH}}{\equiv} & (C_1 \sqcap D) \sqcup (C_2 \sqcap D) \\
& \equiv & (C_1 \sqcup C_2) \sqcap D \\
& \equiv & C \sqcap D
\end{array}
$$

b.) $C_1 \mid D = \bot$ or $C_2 \mid D = \bot$. W.l.o.g. let $C_1 \mid D = \bot$. Then

$$
\begin{array}{lll}
(C \mid D) \sqcap D & \overset{\text{Def. 3.2.31}}{=} & (C_2 \mid D) \sqcap D \\
& \overset{\text{IH}}{\equiv} & C_2 \sqcap D \\
& \equiv & \bot \sqcup (C_2 \sqcap D) \\
& \equiv & (\bot \sqcap D) \sqcup (C_2 \sqcap D) \\
& \equiv & ((C_1 \mid D) \sqcap D) \sqcup (C_2 \sqcap D) \\
& \overset{\text{IH}}{\equiv} & (C_1 \sqcap D) \sqcup (C_2 \sqcap D) \\
& \equiv & (C_1 \sqcup C_2) \sqcap D \\
& \equiv & C \sqcap D
\end{array}
$$

c.) Otherwise:

$$
\begin{array}{lll}
C \mid D \sqcap D & \overset{\text{Def. 3.2.31}}{=} & ((C_1 \mid D) \sqcup (C_2 \mid D)) \sqcap D \\
& \equiv & ((C_1 \mid D) \sqcap D) \sqcup ((C_2 \mid D) \sqcap D) \\
& \overset{\text{IH}}{\equiv} & (C_1 \sqcap D) \sqcup (C_2 \sqcap D) \\
& \equiv & (C_1 \sqcup C_2) \sqcap D \\
& \equiv & C \sqcap D
\end{array}
$$

3.) $C = \forall R.C_1$

a.) Let D be $\exists R.B' \sqcap D_1 \sqcap \ldots \sqcap D_n$ with $C_1 \mid B' = \bot$. Let further D' be $D_1 \sqcap \ldots \sqcap D_n$. According to the induction hypothesis, $(C_1 \mid B') \sqcap B' = C_1 \sqcap B' = \bot$.
Then

$$
\begin{array}{lll}
(C \mid D) \sqcap D & \overset{\text{Def. 3.2.31}}{=} & \bot \sqcap D \\
& \equiv & \bot \\
& \equiv & \forall R.C_1 \sqcap \exists R.\bot \sqcap D' \\
& \equiv & \forall R.C_1 \sqcap \exists R.((C_1 \mid B') \sqcap B') \sqcap D' \\
& \overset{\text{IH}}{\equiv} & \forall R.C_1 \sqcap \exists R.(C_1 \sqcap B') \sqcap D' \\
& \equiv & \forall R.C_1 \sqcap \exists R.B' \sqcap D' \\
& \equiv & \forall R.C_1 \sqcap D \\
& \equiv & C \sqcap D
\end{array}
$$

b.) Let D be $\forall R.B \sqcap D_1 \sqcap \ldots \sqcap D_n$ and there is no D_j with $D_j = \exists R.B'$, with $C_1 \mid B' = \bot$.

Then

$$
\begin{aligned}
(C \mid D) \sqcap D &\overset{\text{Def. 3.2.31}}{=} (\forall R.C_1 \mid B) \sqcap D \\
&\equiv (\forall R.C_1 \mid B) \sqcap \forall R.B \sqcap D_1 \sqcap \ldots \sqcap D_n \\
&\equiv (\forall R.(C_1 \mid B) \sqcap B) \sqcap D \\
&\overset{\text{IH}}{\equiv} \forall R.(C_1 \sqcap B) \sqcap D \\
&\equiv \forall R.C_1 \sqcap \forall R.B \sqcap D_1 \sqcap \ldots \sqcap D_n \\
&\equiv \forall R.C_1 \sqcap D \\
&\equiv C \sqcap D
\end{aligned}
$$

c.) Otherwise:

$$C \mid D \sqcap D \overset{\text{Def. 3.2.31}}{=} \forall R.C_1 \sqcap D = C \sqcap D.$$

4.) $C = \exists R.C_1$:

a.) Let D be $\forall R.B \sqcap D_1 \sqcap \ldots \sqcap D_n$ and

$$C_1 \mid B = \bot. \tag{3.6}$$

Let further D' be $D_1 \sqcap \ldots \sqcap D_n$. Then

$$
\begin{aligned}
(C \mid D) \sqcap D &\overset{\text{Def. 3.2.31}}{=} \bot \sqcap D \\
&\equiv \exists R.\bot \sqcap D \\
&\equiv \exists R.(\bot \sqcap B) \sqcap D \\
&\overset{(3.6)}{\equiv} \exists R.((C_1 \mid B) \sqcap B) \sqcap D \\
&\overset{\text{IH}}{\equiv} \exists R.(C_1 \sqcap B) \sqcap D \\
&\equiv \exists R.C_1 \sqcap \forall R.B \sqcap D' \\
&\equiv \exists R.C_1 \sqcap D \\
&\equiv C \sqcap D
\end{aligned}
$$

b.) Let D be $\forall R.B \sqcap D_1 \sqcap \ldots \sqcap D_n$ and $C_1 \mid B \neq \bot$. Let further D' be $D' = D_1 \sqcap D_n$. Then

$$
\begin{aligned}
(C \mid D) \sqcap D &\overset{\text{Def. 3.2.31}}{=} \exists R.(C_1 \mid B) \sqcap \forall R.B \sqcap D' \\
&\equiv \exists R.((C_1 \mid B) \sqcap B) \sqcap D \\
&\overset{\text{IH}}{\equiv} \exists R.(C_1 \sqcap B) \sqcap D \\
&\equiv \exists R.(C_1 \sqcap B) \sqcap \forall R.B \sqcap D' \\
&\equiv \exists R.C_1 \sqcap \forall R.B \sqcap D' \\
&\equiv C \sqcap D
\end{aligned}
$$

c.) Otherwise: $(C \mid D) \sqcap D \overset{\text{Def. 3.2.31}}{=} \exists R.C_1 \sqcap D = C \sqcap D.$

□

Lemma 3.2.34. *Let C_1 and C_2 be concepts such that $C_1 \sqcap C_2$ is linkless. Let furthermore D be a q-concept. Then $C_1 \sqcap C_2 \sqcap D$ is unsatisfiable iff $C_1 \sqcap D$ or $C_2 \sqcap D$ is unsatisfiable.*

Please note that in the following proof the notion of a path and the conjunction of the elements of a path will be used interchangeably.

Proof. Let C_1 and C_2 be concepts such that $C_1 \sqcap C_2$ is linkless. Let furthermore D be a q-concept. We show that 1. if $C_1 \sqcap D$ or $C_2 \sqcap D$ is unsatisfiable, $C_1 \sqcap C_2 \sqcap D$ is unsatisfiable and 2. if $C_1 \sqcap C_2 \sqcap D$ is unsatisfiable, then $C_1 \sqcap D$ or $C_2 \sqcap D$ is unsatisfiable.

1. Unsatisfiability of $C_1 \sqcap D$ or $C_2 \sqcap D$ clearly implies unsatisfiability of $C_1 \sqcap C_2 \sqcap D$.

2. can be shown by contradiction: Let $C_1 \sqcap C_2 \sqcap D$ be unsatisfiable and assume both $C_1 \sqcap D$ and $C_2 \sqcap D$ are satisfiable. This implies:

   - There is a path P_1 in $\mathsf{paths}(C_1)$ such that $P_1 \sqcap D$ is satisfiable.
   - There is a path P_2 in $\mathsf{paths}(C_2)$ such that $P_1 \sqcap D$ is satisfiable.

   According to Definition 3.2.2,

   $$\mathsf{paths}(C_1 \sqcap C_2) = \{X \cup Y \mid X \in \mathsf{paths}(C_1) \text{ and } Y \in \mathsf{paths}(C_2)\}.$$

   Therefore $P_1 \cup P_2 \in \mathsf{paths}(C_1 \sqcap C_2)$. Since $C_1 \sqcap C_2$ is linkless, $P_1 \cup P_2$ is satisfiable. We show that this implies the consistency of $(P_1 \sqcup P_2) \sqcap D$. D is a q-concept which implies that $\mathsf{paths}(D)$ only contains one path P_D. According to Lemma 3.2.3 there are four possibilities for $P_1 \cup P_2 \cup P_D$ to be unsatisfiable:

   a) $P_1 \cup P_2 \cup P_D$ contains $\bot$. However since $P_1 \cup P_2$ is satisfiable and since the satisfiable of q-concepts implies that P_D does not contain $\bot$, this case is not possible.

   b) P_D contains a literal concept L and $P_1 \cup P_2$ contains $\overline{L}$. However since $P_1 \cup P_D$ and $P_2 \cup D$ are satisfiable, this case is not possible.

   c) $\exists R.C \in (P_1 \cup P_2 \cup P_D)$ contains where C is unsatisfiable. Since $P_1 \cup P_2$ and P_D are satisfiable, this case is not possible.

   d) $\exists R.C \in (P_1 \cup P_2 \cup P_D)$ and $\{\forall R.D_1, \ldots, \forall R.D_n\} \subseteq (P_1 \cup P_2 \cup P_D)$ with $C \sqcap D_1 \sqcap \ldots D_n$ unsatisfiable. Since $C_1 \sqcap C_2$ is linkless, there can be at most one concept of the form $\forall R.B$ in $P_1 \cup P_2$. We investigate two cases:

      i. P_D contains $\exists R.C$ and $P_1 \cup P_2$ contains $\forall R.B$ with $C \sqcap B$ unsatisfiable. However since $P_1 \cup P_D$ and $P_2 \cup P_D$ are satisfiable, this case is not possible.

      ii. P_D contains $\forall R.C$ and $P_1 \cup P_2$ contains $\exists R.B$ with $B \sqcap C$ unsatisfiable. However since $P_1 \cup P_D$ and $P_2 \cup P_D$ are satisfiable, this case is not possible.

   Since all four cases are not possible, $P_1 \cup P_2 \cup P_D$ has to be satisfiable. This is a contradiction to the assumption that $C_1 \sqcap C_2 \sqcap D$ is unsatisfiable and both $C_1 \sqcap D$ and $C_2 \sqcap D$ are satisfiable and it follows that $C_1 \sqcap D$ or $C_2 \sqcap D$ has to be unsatisfiable.

□

Lemma 3.2.35. *Let C be a linkless concept and D be a q-concept. Then $C \mid D$ is in propagated $\exists$-NF.*

Proof. Let C be a linkless concept and D be a q-concept. We show that $C \mid D$ is in propagated $\exists$-NF. Conditioning does not introduce new role restrictions. Furthermore, conditioning does not introduce new paths. It follows that, since C is in $\exists$-NF, the result of conditioning is in $\exists$-NF as well. It remains to show that the result of conditioning is always in propagated $\exists$-NF. We prove this by considering subconcepts of C of the form $\exists R.A \sqcap \forall R.B$ and show that $(\exists R.A \sqcap \forall R.B) \mid D$ is always in propagated $\exists$-NF.

Since $\exists R.A \sqcap \forall R.B$ is linkless,

$$\exists R.A \sqcap \forall R.B \equiv \exists R.(A \sqcap B) \sqcap \forall R.B.$$

We show $\exists R.(A \sqcap B) \sqcap \forall R.B \mid D$ is in propagated $\exists$-NF by distinguishing different cases for D and applying Definition 3.2.31. Note that due to the structure of $\exists R.(A \sqcap B) \sqcap \forall R.B$, we only have to consider the cases 4. and 5. of Definition 3.2.31. Furthermore, since conditioning does not add new paths to a concept, it is sufficient to only consider occurrences of $\exists R.D_1$ and $\forall R.D_2$ in the q-concept D.

1. D contains $\exists R.D_1$:
   a) $B \mid D_1 = \bot$: Then $(\forall R.B)|D = \bot$ (according to the first case in 4. of Definition 3.2.31) and further
   $$(\exists R.(A \sqcap B) \sqcap \forall R.B) \mid D = \bot$$
   which is in propagated $\exists$-NF.
   b) $B \mid D_1 \neq \bot$:
      i. D does not contain $\forall R.D_2$: Then
      $$(\exists R.(A \sqcap B) \sqcap \forall R.B) \mid D = \exists R.(A \sqcap B) \sqcap \forall R.B$$
      which is in propagated $\exists$-NF.
      ii. D contains $\forall R.D_2$: $\exists R.D_1$ is irrelevant for the result of the conditioning, since $B \mid D_1 \neq \bot$. Therefore
      $$(\exists R.(A \sqcap B) \sqcap \forall R.B) \mid D = (\exists R.(A \sqcap B) \sqcap \forall R.B) \mid \forall R.D_2.$$
      - $(A \sqcap B) \mid D_2 = \bot$: This corresponds to the first case in 5. given in Definition 3.2.31. Then
      $$(\exists R.(A \sqcap B) \sqcap \forall R.B) \mid \forall R.D_2 = \bot$$
      which is in propagated $\exists$-NF.
      - $(A \sqcap B) \mid D_2 \neq \bot$: This corresponds to the second case in 5. given in Definition 3.2.31. This implies, that both $A \mid D_2 \neq \bot$ and $B \mid D_2 \neq \bot$. This leads to:
      $$\begin{aligned} & (\exists R.(A \sqcap B) \sqcap \forall R.B) \mid \forall R.D_2 \\ = \; & \exists R.(A \sqcap B) \mid D_2 \sqcap (\forall R.B) \mid D_2 \\ \overset{\text{Def. 3.2.31}}{=} \; & \exists R.(A \mid D_2 \sqcap B \mid D_2) \sqcap (\forall R.B) \mid D_2 \end{aligned}$$
      which is in propagated $\exists$-NF.

2. D does not contain $\exists R.D_1$:

   a) D does not contain $\forall R.D_2$: This corresponds to the third case in 4. and 5. given in Definition 3.2.31. Then

   $$(\exists R.(A \sqcap B) \sqcap \forall R.B) \mid D = \exists R.(A \sqcap B) \sqcap \forall R.B$$

   which is in propagated $\exists$-NF.

   b) D contains $\forall R.D_2$:

      i. $(A \sqcap B) \mid D_2 = \bot$: This corresponds to the first case in 5. given in Definition 3.2.31. Then

      $$(\exists R.(A \sqcap B) \sqcap \forall R.B) \mid \forall R.D_2 = \bot$$

      which is in propagated $\exists$-NF.

      ii. $(A \sqcap B) \mid D_2 \neq \bot$: This corresponds to the second case in 5. given in Definition 3.2.31. This implies, that both $A \mid D_2 \neq \bot$ and $B \mid D_2 \neq \bot$. This leads to:

      $$\begin{array}{ll} & (\exists R.(A \sqcap B) \sqcap \forall R.B) \mid \forall R.D_2 \\ = & \exists R.(A \sqcap B) \mid D_2 \sqcap (\forall R.B) \mid D_2 \\ \overset{\text{Def. } 3.2.31}{=} & \exists R.(A \mid D_2 \sqcap B \mid D_2) \sqcap (\forall R.B) \mid D_2 \end{array}$$

      which is in propagated $\exists$-NF.

We have shown that the result of $(\exists R.(A \sqcap B) \sqcap \forall R.B) \mid D$ is in propagated $\exists$-NF for all possible D which immediately implies the assertion. □

Lemma 3.2.36. *Let C be a linkless concept and D be a q-concept. Then $C \mid D$ is satisfiable, iff $C \sqcap D$ is satisfiable. Furthermore $C \mid D$ is linkless.*

Proof. Let C be a linkless concept and D be a q-concept. We show that 1. satisfiability of $C \sqcap D$ implies satisfiability of $C \mid D$ and 2. satisfiability of $C \mid D$ implies satisfiability of $C \sqcap D$ and furthermore $C \mid D$ is linkless.

1. follows immediately from Lemma 3.2.33.

2. is shown by induction on the structure of concept C:

   Induction basis: C is a literal concept.

   a) $C \in D$:
   Then $C \mid D = \top$, which is satisfiable. Since all q-concepts are satisfiable and $C \in D$, it follows that $C \sqcap D$ is satisfiable as well. Besides this $C \mid D = \top$ is linkless.

   b) $\overline{C} \in D$:
   Then $C \mid D = \bot = C \sqcap D$. Furthermore $\bot$ is linkless.

c) $\overline{C} \notin D$ and $C \notin D$:

i. $C \neq \bot$: Then $C \mid D = C$, which is satisfiable, since all linkless concepts which are not equal to $\bot$ are satisfiable. Further $C \sqcap D$ is satisfiable, since D is satisfiable by definition and C is a literal concept not occurring in D. Besides this C is linkless.

ii. $C = \bot$: Then $C \mid D = \bot$ which is unsatisfiable. Furthermore $C \sqcap D = \bot \sqcap D = \bot$ is unsatisfiable as well. In addition to that, C is linkless.

Induction hypothesis: For linkless concepts C_1, C_2 and all q-concepts D: If $C_i \mid D$ is satisfiable then $C_i \sqcap D$ is satisfiable as well. Furthermore $C_i \mid D$ is linkless where $i \in \{1, 2\}$.

Sometimes we use the contraposition of the induction hypothesis, namely: If $C_i \sqcap D$ is unsatisfiable, then $C_i \mid D$ is unsatisfiable.

Induction step: We have to show, that the assertion holds for linkless concepts $C_1 \sqcap C_2$, $C_1 \sqcup C_2$, $\forall R.C_1$ and $\exists R.C_1$.

a) $C = C_1 \sqcap C_2$:

i. $C_1 \mid D = \bot$ or $C_2 \mid D = \bot$. W.l.o.g. $C_1 \mid D = \bot$:
Then $C \mid D = \bot$ which is unsatisfiable and linkless. Furthermore, according to the contraposition of 1., $C_1 \sqcap D$ has to be unsatisfiable. This implies the unsatisfiability of $C \sqcap D = C_1 \sqcap C_2 \sqcap D$.

ii. $C_1 \mid D = \top$ or $C_2 \mid D = \top$. W.l.o.g. $C_1 \mid D = \top$.
According to the induction hypothesis $C_1 \sqcap D$ is satisfiable. We assume $C \mid D$ to be satisfiable and show that $C \sqcap D$ is satisfiable as well.
According to Definition 3.2.31, $C \mid D \overset{\text{Def. } 3.2.31}{=} C_2 \mid D$.

A. $C_2 \sqcap D$ unsatisfiable:
From the contraposition of the induction hypothesis follows that $C_2 \mid D$ is unsatisfiable as well. Since $C \mid D \overset{\text{Def. } 3.2.31}{=} C_2 \mid D$, this is a contradiction to the assumption that $C \mid D$ is satisfiable.

B. $C_2 \sqcap D$ satisfiable:
Since $C_1 \sqcap C_2$ is linkless and both $C_1 \sqcap D$ and $C_2 \sqcap D$ are satisfiable, it follows from Lemma 3.2.34 that $C_1 \sqcap C_2 \sqcap D = C \sqcap D$ is satisfiable.

Furthermore, according to the induction hypothesis, $C \mid D = C_2 \mid D$ is linkless.

iii. $C_i \mid D \neq \bot$ and $C_i \mid D \neq \top$ for $i \in \{1, 2\}$:
Then $C \mid D = (C_1 \mid D) \sqcap (C_2 \mid D)$. If $C \mid D$ is satisfiable, both $C_i \mid D$ have to be satisfiable. According to the induction hypothesis, this implies the satisfiability off $C_i \sqcap D$ ($i \in \{1, 2\}$) and thus the satisfiability of $C \sqcap D$.

In addition to that $C \mid D = C_1 \mid D \sqcap C_2 \mid D$ is linkless. This is the case, since $C_1 \sqcap C_2$ is linkless and the conditioning with D does not introduce links. According to Lemma 3.2.35, the result of conditioning is always in propagated $\exists$-NF and therefore $C \mid D$ is linkless.

b) $C = C_1 \sqcup C_2$

i. $C_1 \mid D = \top$ or $C_2 \mid D = \top$. W.l.o.g. $C_1 \mid D = \top$.
Then $C \mid D = (C_1 \sqcup C_2) \mid D \overset{\text{Def. } 3.2.31}{=} \top$ is satisfiable and linkless. According to the induction hypothesis $C_1 \sqcap D$ is satisfiable, which implies $C \sqcap D = (C_1 \sqcup C_2) \sqcap D = (C_1 \sqcap D) \sqcup (C_2 \sqcap D)$ to be satisfiable as well.

ii. $C_1 \mid D = \bot$ or $C_2 \mid D = \bot$. W.l.o.g. $C_1 \mid D = \bot$.
Then $C \mid D \overset{\text{Def. } 3.2.31}{=} C_2 \mid D$. If $C_2 \mid D$ is satisfiable, $C_2 \sqcap D$ has to be satisfiable as well, according to the induction hypothesis. Further

$$\begin{aligned} C \sqcap D &= (C_1 \sqcup C_2) \sqcap D \\ &= (C_1 \sqcap D) \sqcup (C_2 \sqcap D). \end{aligned}$$

To sum up: Consistency of $C \mid D$ implies consistency of $C_2 \sqcap D$ which implies consistency of $C \sqcap D$.

In addition to that, $C \mid D = C_2 \mid D$ is linkless according to the induction hypothesis.

iii. $C_i \mid D \neq \top$ and $C_i \mid D \neq \bot$ for $i \in \{1, 2\}$.
Then $C \mid D \overset{\text{Def. } 3.2.31}{=} (C_1 \mid D) \sqcup (C_2 \mid D)$, which is satisfiable, iff $C_1 \mid D$ or $C_2 \mid D$ is satisfiable. W.l.o.g. let $C_1 \mid D$ be satisfiable. By induction hypothesis, this implies the consistency of $C_1 \sqcap D$, which leads to the consistency of

$$\begin{aligned} C \sqcap D &= (C_1 \sqcup C_2) \sqcap D \\ &\equiv (C_1 \sqcap D) \sqcup (C_2 \sqcap D). \end{aligned}$$

Furthermore both $C1 \mid D$ and $C_2 \mid D$ are linkless according to the induction hypothesis. Hence $C \mid D = C_1 \mid D \sqcup C_2 \mid D$ is linkless as well.

c) $C = \forall R.C_1$

i. Let D be $\exists R.B' \sqcap D_1 \sqcap \ldots \sqcap D_n$ and $D' = D_1 \sqcap \ldots \sqcap D_n$ with $C_1 \mid B' = \bot$.
Then $C \mid D \overset{\text{Def. } 3.2.31}{=} \bot$. Since we have to show that satisfiability of $C \mid D$ implies satisfiability of $C \sqcap D$ we are done with this case.

ii. Let D be $\forall R.B \sqcap D_1 \sqcap \ldots \sqcap D_n$ and $D' = D_1 \sqcap \ldots \sqcap D_n$ and there is no $\exists R.B'$ in D with $C_1 \mid B' = \bot$.
Then $C \mid D \overset{\text{Def. } 3.2.31}{=} \forall R.(C_1 \mid B)$ which is satisfiable and according to the induction hypothesis is linkless. Further

$$\begin{aligned} C \sqcap D &= \forall R.C_1 \sqcap D \\ &= \forall R.C_1 \sqcap \forall R.B \sqcap D' \\ &\equiv \forall R.(C_1 \sqcap B) \sqcap D. \end{aligned}$$

This is satisfiable, since we claimed that there is no $\exists R.B'$ in D with $C_1 \mid B' = \bot$.

iii. $\forall R.B \notin D$ and there is no $\exists R.B'$ in D with $C_1 \mid B' = \bot$.
Then $C \mid D \overset{\text{Def. } 3.2.31}{=} \forall R.C_1$, which is satisfiable. Furthermore, according to the induction hypothesis, $C \mid D \overset{\text{Def. } 3.2.31}{=} \forall R.C_1$ is linkless. Further $C \sqcap D = \forall R.C_1 \sqcap D$ has to be satisfiable, since D is a q-concept and for all existential role restriction $\exists R.B_i$ in D holds $C_1 \mid B_i \neq \bot$ which implies the consistency of $C_1 \sqcap B_i$.

d) $C = \exists R.C_1$

i. Let D be $\forall R.B \sqcap D_1 \sqcap \ldots \sqcap D_n$, $D' = D_1 \sqcap \ldots \sqcap D_n$ and $C_1 \mid B = \bot$. It follows that $C \mid D \overset{\text{Def. } 3.2.31}{=} \bot$, which is unsatisfiable and linkless. Since we have to show that satisfiability of $C \mid D$ implies satisfiability of $C \sqcap D$ we are done with this case.

ii. Let $\forall R.B \in D$ and $C_1 \mid B \neq \bot$.
Then $C \mid D \overset{\text{Def. } 3.2.31}{=} \exists R.(C_1 \mid B)$ is satisfiable. Further it follows from the induction hypothesis, that $C_1 \sqcap B$ is satisfiable.
Let D be $\forall R.B \sqcap D_1 \sqcap \ldots \sqcap D_n$ and $D' = D_1 \sqcap \ldots \sqcap D_n$. Since D is a q-concept, it is ensured that no D_j in D' has the form $\forall R.B'$.Then

$$\begin{aligned} C \sqcap D &= \exists R.C_1 \sqcap D \\ &= \exists R.C_1 \sqcap \forall R.B \sqcap D' \\ &\equiv \exists R.(C_1 \sqcap B) \sqcap D. \end{aligned}$$

Since both $C_1 \sqcap B$ and D are satisfiable and further $\forall R.B' \notin D$ for all $B' \neq B$, it follows that $C \sqcap D$ is satisfiable.
From the induction hypothesis follows, that $C_1 \mid B$ is linkless. Hence $C \mid D = \exists R.(C_1 \mid B)$ is linkless as well.

iii. Let $\forall R.B \notin D$.
Then $C \mid D \overset{\text{Def. } 3.2.31}{=} \exists R.C_1$, which is satisfiable, iff C_1 is satisfiable.
Further $C \sqcap D = \exists R.C_1 \sqcap D$ is satisfiable, iff C_1 is satisfiable. This follows from the fact that the q-concept D is satisfiable and further $\forall R.B \notin D$.
In addition to that, $C \mid D = \exists R.C_1$ is linkless, because C_1 is linkless according to the induction hypothesis.

□

The next theorem shows how conditioning can be used to perform subsumption checks.

Theorem 3.2.37. *Let C be a linkless concept and D be a q-concept. Then $C \sqsubseteq \neg D$ holds, iff $C \mid D$ is unsatisfiable.*

Proof. Let C be a linkless concept and D be a query concept. $C \sqsubseteq \neg D$ holds iff $C \sqcap D$ is unsatisfiable. According to Lemma 3.2.36, this is the case iff $C \mid D$ is unsatisfiable. □

Since conditioning a linkless concept C with a q-concept D can be performed in time linear in $\mathsf{size}(C) \cdot \mathsf{size}(D)$, checking the above mentioned subsumption $C \sqsubseteq \neg D$ has the same time complexity.

Example 3.2.38. *Consider the following linkless concept:*

$$C = \big(\exists R.(E \sqcap \neg B) \sqcap \forall R.\neg B \sqcap (E \sqcup D)\big) \sqcup \big(\exists R.(E \sqcap \neg B \sqcap F) \sqcap \forall R.(\neg B \sqcap F)\big)$$

We now want to check the subsumption

$$C \sqsubseteq E \sqcup \exists R.F.$$

Negating the right side of the subsumption, leads to the q-concept $\neg E \sqcap \forall R.\neg F$. *With the help of Definition 3.2.31, it is possible compute the result of conditioning* C *by the q-concept:*

$$C \mid \neg E \sqcap \forall R.\neg F = \exists R.(E \sqcap \neg B) \sqcap \forall R.\neg B \sqcap D$$

Since this concept is satisfiable, the subsumption $\models C \sqsubseteq E \sqcup \exists R.F$ *does not hold.*

Uniform Interpolation Section 3.1 introduces projection as an interesting transformation for knowledge compilation target languages for propositional logic. As mentioned before, projection is dual to the notion of forgetting. In the context of description logic, usually the term *uniform interpolation* is used to denote forgetting. Uniform interpolation has many applications with regard to ontologies (Konev, Walther, and Wolter, 2009). One examples is re-use of ontologies, where only a small subset of the vocabulary of an ontology under consideration is of interest. Another interesting application is predicate hiding, where certain parts of an ontology are not supposed to be seen by the public and are therefore hidden. Intuitively, the uniform interpolant of a concept C w.r.t. a set of atomic concept symbols Φ is the concept D, which does not contain any atomic symbols from Φ and is indistinguishable from C w.r.t the consequences that do not use symbols from Φ.

Definition 3.2.39 (Uniform Interpolant)**.** *Let* C *be a concept and* Φ *a set of atomic concepts. Concept* D *is called* uniform interpolant *of* C *w.r.t.* Φ *or short* Φ*-interpolant of* C *iff the following conditions are fulfilled:*

1. D *only contains atomic concepts which occur in* C *but not in* Φ.
2. $\models C \sqsubseteq D$.
3. *For all concepts* E *not containing symbols from* Φ *holds:* $\models C \sqsubseteq E$ *iff* $\models D \sqsubseteq E$.

It can be shown that Φ-interpolants are unique up to equivalence. An algorithm to compute the uniform interpolant of a concept w.r.t. a set of concept symbols Φ was presented by ten Cate, Conradie, Marx, and Venema (2006). The following definition presents an operator to compute the uniform interpolant of a linkless concept w.r.t. a set of atomic concepts.

Definition 3.2.40 (UI-operator)**.** *Let $\Sigma = (N_C, N_R, N_I)$ be a description logic signature, $C \in C^\Sigma$ be a linkless concept and $\Phi \subseteq N_C$ be a set of atomic concepts. Then $\mathsf{UI}(C, \Phi)$ is the concept obtained by substituting each occurrence of the literal concepts A and $\neg A$ in C by $\top$ iff $A \in \Phi$.*

From the way $\mathsf{UI}(C, \Phi)$ is constructed, the following lemma is obvious:

Lemma 3.2.41. *Let $\Sigma = (N_C, N_R, N_I)$ be a description logic signature, $C, C_1, C_2 \in C^\Sigma$ be linkless concepts and $\Phi, \Phi_1, \Phi_2 \subseteq N_C$ be sets of atomic concepts. Then*

1. *If C does not contain any symbols from Φ, then $C = \mathsf{UI}(C, \Phi)$.*
2. $\mathsf{UI}(C_1 \sqcap C_2, \Phi) = \mathsf{UI}(C_1, \Phi) \sqcap \mathsf{UI}(C_2, \Phi)$.
3. $\mathsf{UI}(C_1 \sqcup C_2, \Phi) = \mathsf{UI}(C_1, \Phi) \sqcup \mathsf{UI}(C_2, \Phi)$.
4. $\mathsf{UI}(QR.C, \Phi) = QR.\mathsf{UI}(C, \Phi)$, *for* $Q \in \{\exists, \forall\}$.
5. $\mathsf{UI}(C, \Phi_1 \cup \Phi_2) = \mathsf{UI}(\mathsf{UI}(C, \Phi_1), \Phi_2) = \mathsf{UI}(\mathsf{UI}(C, \Phi_2), \Phi_1)$.

Lemma 3.2.42. *Let $\Sigma = (N_C, N_R, N_I)$ be a description logic signature, $C \in C^\Sigma$ be a linkless concept and $\Phi \subseteq N_C$ be a set of atomic concepts. Then C is satisfiable, iff $\mathsf{UI}(C, \Phi)$ is satisfiable.*

Proof. Let $\Sigma = (N_C, N_R, N_I)$ be a description logic signature, $C \in C^\Sigma$ be a linkless concept and $\Phi \subseteq N_C$ be a set of atomic concepts. We show that 1. satisfiability of C implies satisfiability of $\mathsf{UI}(C, \Phi)$ and 2. satisfiability of $\mathsf{UI}(C, \Phi)$ implies satisfiability of C.

1. we assume that C is satisfiable. Then there has to be a satisfiable path in C. During the construction of $\mathsf{UI}(C, \Phi)$, the only thing that is done are substitutions by $\top$. This can never make a satisfiable path unsatisfiable. Therefore there has to be a satisfiable path in $\mathsf{UI}(C, \Phi)$ as well, which makes $\mathsf{UI}(C, \Phi)$ satisfiable.
2. can be shown by contraposition: We assume that C is unsatisfiable. Since C is linkless, the only way that C is unsatisfiable is that $C = \bot$. Therefore $\mathsf{UI}(C, \Phi) = \mathsf{UI}(\bot, \Phi) = \bot$ for all Φ, which is unsatisfiable.

□

The following proposition states that the UI-function preserves models.

Proposition 3.2.43. *Let $\Sigma = (N_C, N_R, N_I)$ be a description logic signature, $C \in C^\Sigma$ be a linkless concept and $\Phi \subseteq N_C$ be a set of atomic concepts. If $\mathcal{I}$ is a model for C then $\mathcal{I}$ is a model for $\mathsf{UI}(C, \Phi)$ as well.*

Proof. Let $\Sigma = (N_C, N_R, N_I)$ be a description logic signature, $C \in C^\Sigma$ be a linkless concept and $\Phi \subseteq N_C$ be a set of atomic concepts. If Φ is a set with more than one element, the UI-function can be performed step-by-step according to 5. of Lemma 3.2.41.

This is why it is sufficient to show the proposition by assuming that Φ contains a single atomic concept. From this the assertion immediately follows. We show the following assertion by induction on the structure of C:
For all $b \in \Delta^{\mathcal{I}}$, if $b \in C^{\mathcal{I}}$ then $b \in (\mathsf{UI}(C, \Phi))^{\mathcal{I}}$.
This assertion implies that $\mathcal{I}$ is a model for $\mathsf{UI}(C, \Phi)$.

Induction basis: C is a literal concept, $\top$ or $\bot$:

1. $C = A$ or $C = \neg A$ and $A \in \Phi$: Then $\mathsf{UI}(C, \Phi) = \top$. Since $b \in C^{\mathcal{I}}$, $b \in \top^{\mathcal{I}}$ as well, which implies $b \in (\mathsf{UI}(C, \Phi))^{\mathcal{I}}$.

2. $C = A$ or $C = \neg A$ and $A \notin \Phi$: Then $\mathsf{UI}(C, \Phi) = C$ and therefore the assertion immediately follows.

3. $C = \bot$: Since C is unsatisfiable, there is no $b \in \Delta^{\mathcal{I}}$ with $b \in C^{\mathcal{I}}$ and therefore the assertion follows.

4. $C = \top$: Then $\mathsf{UI}(C, \Phi) = \top$. Since $a \in C^{\mathcal{I}}$, $b \in \top^{\mathcal{I}}$ as well, which implies $b \in (\mathsf{UI}(C, \Phi))^{\mathcal{I}}$.

Induction hypothesis: For linkless concepts C_1, C_2, Φ a set consisting of one atomic concept, a model $\mathcal{I}$ for C_i and $a \in \Delta^{\mathcal{I}}$: If $a \in C_i^{\mathcal{I}}$ then $a \in (\mathsf{UI}(C_i, \Phi))^{\mathcal{I}}$, where $i \in \{1, 2\}$.

Induction step:

1. $C = C_1 \sqcap C_2$ and $\mathcal{I}$ is a model for C. Therefore there is $b \in \Delta^{\mathcal{I}}$ with $b \in C^{\mathcal{I}}$. We have to show for every $b \in \Delta^{\mathcal{I}}$ that $b \in (\mathsf{UI}(C, \Phi))^{\mathcal{I}}$ as well.
   $b \in C^{\mathcal{I}}$ and $C = C_1 \sqcap C_2$ means that $b \in (C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}})$. Therefore we have both

   $$b \in C_1^{\mathcal{I}} \text{ and}$$
   $$b \in C_2^{\mathcal{I}}.$$

   From the induction hypothesis it follows

   $$b \in \mathsf{UI}(C_1, \Phi)^{\mathcal{I}} \text{ and}$$
   $$b \in \mathsf{UI}(C_2, \Phi)^{\mathcal{I}}.$$

   Leading to $b \in (\mathsf{UI}(C_1, \Phi)^{\mathcal{I}} \cap \mathsf{UI}(C_2, \Phi)^{\mathcal{I}})$. Furthermore

   $$\begin{aligned} &\mathsf{UI}(C_1, \Phi)^{\mathcal{I}} \cap \mathsf{UI}(C_2, \Phi)^{\mathcal{I}} \\ &= (\mathsf{UI}(C_1, \Phi) \sqcap \mathsf{UI}(C_2, \Phi))^{\mathcal{I}} \\ &= (\mathsf{UI}(C_1 \sqcap C_2, \Phi))^{\mathcal{I}} \\ &= (\mathsf{UI}(C, \Phi))^{\mathcal{I}}. \end{aligned}$$

   It follows that $b \in (\mathsf{UI}(C, \Phi))^{\mathcal{I}}$.

2. $C = C_1 \sqcup C_2$ and $\mathcal{I}$ is a model for C. Therefore there is $b \in \Delta^{\mathcal{I}}$ with $b \in C^{\mathcal{I}}$. For every $b \in C^{\mathcal{I}}$ we have to show $b \in (\mathsf{UI}(C, \Phi))^{\mathcal{I}}$ as well.
$b \in C^{\mathcal{I}}$ and $C = C_1 \sqcup C_2$ means that $b \in (C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}})$. W.l.o.g. $b \in C_1^{\mathcal{I}}$.

   From the induction hypothesis it follows

$$b \in \mathsf{UI}(C_1, \Phi)^{\mathcal{I}}.$$

   It follows that $b \in (\mathsf{UI}(C_1, \Phi) \sqcup \mathsf{UI}(C_2, \Phi))^{\mathcal{I}}$. With Lemma 3.2.41 follows

$$b \in \mathsf{UI}(C_1 \sqcup C_2, \Phi)^{\mathcal{I}}.$$

   Leading to

$$b \in \mathsf{UI}(C, \Phi)^{\mathcal{I}}.$$

3. $C = \exists R.C_1$ and $\mathcal{I}$ is a model for C. Therefore there is $b \in \Delta^{\mathcal{I}}$ with $b \in C^{\mathcal{I}}$. For every $b \in C^{\mathcal{I}}$ we have to show $b \in (\mathsf{UI}(C, \Phi))^{\mathcal{I}}$ as well.
$b \in C^{\mathcal{I}}$ and $C = \exists R.C_1$ means that there is $c \in \Delta^{\mathcal{I}}$ with $(b, c) \in R^{\mathcal{I}}$ and $c \in C_1^{\mathcal{I}}$.

   From the induction hypothesis it follows

$$c \in \mathsf{UI}(C_1, \Phi)^{\mathcal{I}}.$$

   This, together with $(b, c) \in R^{\mathcal{I}}$ implies

$$b \in (\exists R.\mathsf{UI}(C_1, \Phi))^{\mathcal{I}}.$$

   It immediately follows from Lemma 3.2.41 that

$$b \in (\mathsf{UI}(\exists R.C_1, \Phi))^{\mathcal{I}}.$$

   Therefore $b \in (\mathsf{UI}(C, \Phi))^{\mathcal{I}}$.

4. $C = \forall R.C_1$ and $\mathcal{I}$ is a model for C. Therefore there is $b \in \Delta^{\mathcal{I}}$ with $b \in C^{\mathcal{I}}$. We have to show $b \in (\mathsf{UI}(C, \Phi))^{\mathcal{I}}$ as well.
$b \in C^{\mathcal{I}}$ and $C = \forall R.C_1$ means that two cases have to be distinguished:
   a) There is no $c \in \Delta^{\mathcal{I}}$ with $(b, c) \in R^{\mathcal{I}}$.
   Then trivially $b \in (\forall R.\mathsf{UI}(C_1, \Phi))^{\mathcal{I}}$. According to Lemma 3.2.41 this implies

$$b \in \mathsf{UI}(\forall R.C_1, \Phi)^{\mathcal{I}}.$$

   Leading to

$$b \in (\mathsf{UI}(C, \Phi))^{\mathcal{I}}.$$

   b) There is a $c \in \Delta^{\mathcal{I}}$ with $(b, c) \in R^{\mathcal{I}}$.
   Since $\mathcal{I}$ is a model for $\forall R.C_1$, for all these individuals c, it follows that $c \in C_1^{\mathcal{I}}$.
   According to the induction hypothesis for all these individuals c,

$$c \in (\mathsf{UI}(C_1, \Phi))^{\mathcal{I}}.$$

Therefore,

$$b \in (\forall R.\mathsf{UI}(C_1, \Phi))^{\mathcal{I}}.$$

From Lemma 3.2.41 it follows that

$$b \in (\mathsf{UI}(\forall R.C_1, \Phi))^{\mathcal{I}}.$$

Leading to

$$b \in (\mathsf{UI}(C, \Phi))^{\mathcal{I}}.$$

□

The next theorem states that the uniform interpolant of a linkless concept w.r.t. a set of concept symbols can be computed efficiently using the UI operator.

Theorem 3.2.44. *Let $\Sigma = (N_C, N_R, N_I)$ be a description logic signature, $C \in C^{\Sigma}$ be a linkless concept and $\Phi \subseteq N_C$ be a set of atomic concepts. Then*

1. *$\mathsf{UI}(C, \Phi)$ is the Φ-interpolant of C,*
2. *$\mathsf{UI}(C, \Phi)$ can be computed in time linear in the size of C and*
3. *if $\mathsf{UI}(C, \Phi)$ is simplified according to the equivalences in Figure 3.2, then $\mathsf{UI}(C, \Phi)$ is linkless.*

Proof. Let $\Sigma = (N_C, N_R, N_I)$ be a description logic signature, $C \in C^{\Sigma}$ be a linkless concept and $\Phi \subseteq N_C$ be a set of atomic concepts. The second and the third assertion follow directly from the way, $\mathsf{UI}(C, \Phi)$ is constructed. To proof the first assertion of Theorem 3.2.44, namely that $\mathsf{UI}(C, \Phi)$ is the Φ-interpolant of C, we have to show that $\mathsf{UI}(C, \Phi)$ has the three properties given in Definition 3.2.39.

1. The property that $\mathsf{UI}(C, \Phi)$ contains only atomic concept symbols which occur in C but not in Φ follows directly from the way $\mathsf{UI}(C, \Phi)$ is constructed.

2. The property, that $\models C \sqsubseteq \mathsf{UI}(C, \Phi)$ means, that $C \sqcap \neg\mathsf{UI}(C, \Phi)$ is unsatisfiable, which follows directly from Proposition 3.2.43.

3. The third property is that, for all concepts E not containing symbols from Φ holds: $\models C \sqsubseteq E$ iff $\models \mathsf{UI}(C, \Phi) \sqsubseteq E$. We prove this by showing that for all concepts E not containing symbols from Φ, $C \sqcap \neg E$ is unsatisfiable iff $\mathsf{UI}(C, \Phi) \sqcap \neg E$ is unsatisfiable.

   First note that, if C or $\neg E$ is unsatisfiable, then $C \sqcap \neg E$ and $\mathsf{UI}(C, \Phi) \sqcap \neg E$ are unsatisfiable as well and the assertion holds. Therefore we assume C and $\neg E$ to be satisfiable.

   a) If $C \sqcap \neg E$ is satisfiable, then there is a satisfiable path P in $C \sqcap \mathsf{nnf}(\neg E)$. We show that this path P corresponds to a satisfiable path P' in $\mathsf{UI}(C, \Phi) \sqcap \mathsf{nnf}(\neg E)$. Let P_C denote the subpath of P, passing through C and $P_{\neg E}$ be the subpath of P passing through $\mathsf{nnf}(\neg E)$. If we consider $\mathsf{UI}(C, \Phi) \sqcap \neg E$, we know

that the construction of $\mathsf{UI}(C, \Phi)$ only changes the subpath P_C of P such that each occurrence of A and $\neg A$ with $A \in \Phi$ is substituted by $\top$. This can not cause path P to become unsatisfiable, so the resulting path P' is satisfiable as well. Therefore $\mathsf{UI}(C, \Phi) \sqcap \mathsf{nnf}(\neg E)$ and $\mathsf{UI}(C, \Phi) \sqcap \neg E$ is satisfiable.

b) If $C \sqcap \neg E$ is unsatisfiable, then all paths in $C \sqcap \mathsf{nnf}(\neg E)$ are unsatisfiable. However due to the assumption that both C and $\neg E$ are satisfiable, we know that the subpaths P_C and $P_{\neg E}$ are satisfiable. Therefore the contradiction in a path has to be constructed from both elements from P_C and $P_{\neg E}$ and has to use symbols not occurring in Φ. However these symbols are not affected by the construction of UI. Thus the contradictions are still contained in the paths of $\mathsf{UI}(C, \Phi) \sqcap \mathsf{nnf}(\neg E)$, which implies that $\mathsf{UI}(C, \Phi) \sqcap \neg E$ has to be unsatisfiable.

□

Example 3.2.45. *Consider the following linkless concept*

$$C = \big(\exists R.(E \sqcap \neg B) \sqcap \forall R.\neg B \sqcap (E \sqcup D)\big) \sqcup \big(\exists R.(E \sqcap \neg B \sqcap F) \sqcap \forall R.(\neg B \sqcap F)\big)$$

together with set $\Phi = \{E, D\}$. *The* Φ*-interpolant of* C *can be computed using the* UI*-operator leading to:*

$$\mathsf{UI}(C, \Phi) = (\exists R.(\top \sqcap \neg B) \sqcap \forall R.\neg B \sqcap (\top \sqcup \top)) \sqcup (\exists R.(\top \sqcap \neg B \sqcap F) \sqcap \forall R.(\neg B \sqcap F)).$$

$\mathsf{UI}(C, \Phi)$ *can be simplified according to the equivalences in Figure 3.2 to the concept*

$$(\exists R.\neg B \sqcap \forall R.\neg B) \sqcup (\exists R.(\neg B \sqcap F) \sqcap \forall R.(\neg B \sqcap F)).$$

To sum up, Theorem 3.2.44 shows that the UI-function can be used to determine uniform interpolants for a linkless concept and a set of atomic concepts. Furthermore, this computation can be performed in time linear in the size of the linkless concept.

Corollary 3.2.46. *Let* C *be a linkless concept and* Φ *be a set of atomic concepts. Then the* Φ*-interpolant of* C *can be computed in time linear to* $\mathsf{size}(C)$.

Proof. Corollary 3.2.46 immediately follows from Theorem 3.2.44. □

Since the result of the UI-function is linkless, it is possible to perform efficient queries and transformation with the uniform interpolant.

Next, linkless normal form is extended for TBoxes.

3.2.3 Linkless Normal Form for $\mathcal{ALC}$ TBoxes

Until now the linkless normal form is only defined for concepts. However in description logics, the TBox plays an important role as well. For example it is of interest to test if a certain subsumption holds w.r.t. a given TBox. Therefore it is desirable to have a normal form for the TBox as well which allows us to perform certain queries and transformations

efficiently. In order to extend the linkless NF for TBoxes, we first have to consider TBox approximations. According to Horrocks (2003) it is possible to transform a TBox

$$\mathcal{T} = \{A_1 \sqsubseteq B_1, \ldots, A_n \sqsubseteq B_n\}$$

into a metaconstraint by first transforming each axiom

$$A_i \sqsubseteq B_i$$

into an equivalent axiom

$$\top \sqsubseteq \neg A_i \sqcup B_i.$$

These axioms

$$\begin{aligned} &\top \sqsubseteq \neg A_1 \sqcup B_1 \\ &\top \sqsubseteq \neg A_2 \sqcup B_2 \\ &\quad\vdots \\ &\top \sqsubseteq \neg A_n \sqcup B_n \end{aligned}$$

can be conjunctively combined into one axiom

$$\top \sqsubseteq (\neg A_1 \sqcup B_1) \sqcap \ldots \sqcap (\neg A_n \sqcup B_n)$$

which is abbreviated as

$$\top \sqsubseteq C_{\mathcal{T}}$$

with

$$C_{\mathcal{T}} = \bigsqcap_{A_i \sqsubseteq B_i \in \mathcal{T}} (\neg A_i \sqcup B_i).$$

This axiom states that every element in the domain has to belong to the concept $C_{\mathcal{T}}$. $C_{\mathcal{T}}$ is often called metaconstraint. Now subsumptions w.r.t. the TBox $\mathcal{T}$ can be checked using $C_{\mathcal{T}}$. For example testing satisfiability of a concept D w.r.t. $\mathcal{T}$ can be done by checking the satisfiability of $D \sqcap C_{\mathcal{T}} \sqcap \forall U.C_{\mathcal{T}}$. Where U is the transitive closure of the union of all roles occurring in $\mathcal{T}$. $C_{\mathcal{T}} \sqcap \forall U.C_{\mathcal{T}}$ is not an $\mathcal{ALC}$ concept, which is the reason why we use an approximation of this concept.

Definition 3.2.47 (n-th Approximation of a TBox (Wang et al., 2009))**.** *Let* $\Sigma = (N_C, N_R, N_I)$ *be a description logic signature,* $\mathcal{T}$ *be a TBox over* Σ *and* $n \in \mathbb{N}$, $n \geq 0$. *The* n-th approximation of $\mathcal{T}$ *is defined as:*

$$C_{\mathcal{T}}^{(n)} = \bigsqcap_{k=0}^{n} \bigsqcap_{R_1,\ldots,R_k \in N_R} \forall R_1 \ldots \forall R_k.C_{\mathcal{T}}$$

where N_R *the set of all roles occurring in* $\mathcal{T}$ *and* $C_{\mathcal{T}} = \bigsqcap_{A \sqsubseteq B \in \mathcal{T}} (\neg A \sqcup B)$.

From the way $C_{\mathcal{T}}^{(n)}$ is constructed, it follows that

$$C_{\mathcal{T}}^{(n)} = C_{\mathcal{T}}^{(n-1)} \sqcap \prod_{R_1,\ldots,R_n \in \mathcal{R}} \forall R_1 \ldots \forall R_n . C_{\mathcal{T}}$$

which can be used to construct $C_{\mathcal{T}}^{(n)}$ in an iterative way.

Example 3.2.48. *Consider the TBox*

$$\mathcal{T} = \{A \sqsubseteq (B \sqcup \exists R.D),\ B \sqsubseteq \forall R'.D\}.$$

Constructing the n-th approximation for $n \in \{0, 1, 2\}$ *leads to*

$$\begin{aligned}
C_{\mathcal{T}}^{(0)} &= C_{\mathcal{T}} = (\neg A \sqcup B \sqcup \exists R.D) \sqcap (\neg B \sqcup \forall R'.D) \\
C_{\mathcal{T}}^{(1)} &= C_{\mathcal{T}}^{(0)} \sqcap \forall R.C_{\mathcal{T}} \sqcap \forall R'.C_{\mathcal{T}} \\
C_{\mathcal{T}}^{(2)} &= C_{\mathcal{T}}^{(1)} \sqcap \forall R.\forall R.C_{\mathcal{T}} \sqcap \forall R.\forall R'.C_{\mathcal{T}} \sqcap \forall R'.\forall R.C_{\mathcal{T}} \sqcap \forall R'.\forall R'.C_{\mathcal{T}}.
\end{aligned}$$

Next we describe how to use the n-the approximation of a TBox to check if a concept is entailed by the TBox. For this, we present a Lemma, which was first presented by ten Cate et al. (2006). We slightly adapt the lemma to fit our notation.

Lemma 3.2.49. *(ten Cate et al., 2006) Let* C_1, C_2 *be concepts,* $\mathcal{T}$ *be a TBox and* N_R *the set of roles occuring in* C_1, C_2 *and* $\mathcal{T}$*. Then*

$$C_1 \sqsubseteq_{\mathcal{T}} C_2 \quad \textit{iff} \quad \Big(C_1 \sqcap \prod_{\substack{R_1,\ldots,R_n \in N_R, \\ n \leq 2^{\mathsf{size}(\mathcal{T})+\mathsf{size}(C_1)+\mathsf{size}(C_2)}}} \forall R_1 \ldots \forall R_n C_{\mathcal{T}}\Big) \sqsubseteq C_2.$$

From Lemma 3.2.49 it follows that we can use the approximation $C_{\mathcal{T}}^{(n)}$ of a TBox $\mathcal{T}$ to check the satisfiability of a concpet w.r.t. $\mathcal{T}$.

Theorem 3.2.50. *Let* $\mathcal{T}$ *be a TBox,* D *a concept and* $n \in \mathbb{N}$*. If* $n \geq 2^{\mathsf{size}(D)+\mathsf{size}(\mathcal{T})+1}$*, then* D *is satisfiable w.r.t* $\mathcal{T}$ *iff* $D \sqcap C_{\mathcal{T}}^{(n)}$ *is satisfiable.*

Proof. The theorem follows directly from Lemma 3.2.49 with $C_1 = \top$ and $C_2 = D$. □

The notion of the nth-approximation of a TBox can be used to extend the linkless normal form to TBoxes. The idea is to transform $C_{\mathcal{T}}^{(n)}$ into linkless normal form for some number n. This linkless $C_{\mathcal{T}}^{(n)}$ can be used to check satisfiability of q-concepts D w.r.t $\mathcal{T}$. For this, we choose $n \geq 2^{\mathsf{size}(D)+\mathsf{size}(\mathcal{T})+1}$. The higher we choose n the higher the size of the q-concept D can be. It is reasonable to assume that q-concepts are small compared to the size of the TBox. Hence the size of D influences the number $2^{\mathsf{size}(D)+\mathsf{size}(\mathcal{T})+1}$ only to a lesser extent.

Uniform Interpolation The previous section showed how to efficiently compute uniform interpolants for linkless concepts and a set of atomic concepts. For TBoxes, uniform interpolation is even more interesting. In many applications, only a small subset of the signature of a given TBox is used. One application is information hiding, where a part of the signature of the TBox is supposed to be hidden from the user. This leads to the idea of uniform interpolation where all atomic concepts that are not of interest are removed preserving the meaning of the original TBox within the atomic concepts of interest.

Definition 3.2.51 (Φ-Interpolant of a TBox)**.** *Let $\mathcal{T}$ be a TBox and Φ a set of atomic concepts. Then the TBox $\mathcal{T}'$ is called a* uniform interpolant of $\mathcal{T}$ w.r.t. Φ *or short* Φ-interpolant of $\mathcal{T}$ *iff the following conditions hold:*

- *$\mathcal{T}'$ contains only atomic concepts occurring in $\mathcal{T}$ but not in Φ.*
- *$\mathcal{T} \models \mathcal{T}'$.*
- *For all concept inclusions $C \sqsubseteq D$ not containing symbols from Φ: $\mathcal{T} \models C \sqsubseteq D$ implies $\mathcal{T}' \models C \sqsubseteq D$.*

As shown in Wang et al. (2009), there are $\mathcal{ALC}$ TBoxes $\mathcal{T}$ for which the uniform interpolant of $\mathcal{T}$ w.r.t. a set of atomic concepts Φ does not exist. Example 3.2.52, taken from Lutz and Wolter (2011) and slightly modified to our purpose, illustrates uniform interpolation of TBoxes.

Example 3.2.52. *In the following, we present two different TBoxes to illustrate uniform interpolation.*

1. *Let $\mathcal{T}_1$ be a TBox and Φ a set of atomic concepts given as*

$$\begin{aligned} \mathcal{T}_1 &= \{A \sqsubseteq \exists R.B \sqcap \exists R.\neg B\}, \\ \Phi &= \{B\}. \end{aligned}$$

   The Φ-interpolant of $\mathcal{T}_1$ is

$$\mathcal{T}_1' = \{A \sqsubseteq \exists R.\top\}.$$

2. *Let $\mathcal{T}_2$ be a TBox and Φ a set of atomic concepts given as*

$$\begin{aligned} \mathcal{T}_2 = \{&A \sqsubseteq B, \\ &B \sqsubseteq \exists R.B\}, \\ \Phi = \{&B\}. \end{aligned}$$

   $\mathcal{T}_2$ states that starting from every individual belonging to A there has to be an infinite sequence of R-successors, all belonging to the concept B. Without the concept B, this cannot be stated. Therefore there is no Φ-interpolant of $\mathcal{T}_2$.

The afore introduced approximation of TBoxes can be used to reason with respect to the uniform interpolant of TBoxes as well. For this, we partially present a proposition introduced by Wang et al. (2009), adapted to our notaion.

Proposition 3.2.53. *(Wang et al., 2009) Let $\mathcal{T}$ be a TBox, Φ a set of atomic concepts and C_1, C_2 concepts not containing any atomic concepts from Φ. Let further $n \geq 2^{\mathsf{size}(\mathcal{T})+\mathsf{size}(C_1)+\mathsf{size}(C_2)}$ and $\mathcal{T}'$ be the Φ-interpolant of $C_{\mathcal{T}}^{(n)}$. Then $C_1 \sqsubseteq C_2$ w.r.t. $\mathcal{T}$ iff $C_1 \sqsubseteq C_2$ w.r.t. $\mathcal{T}'$.*

From Theorem 3.2.44, Theorem 3.2.50 and Proposition 3.2.53 follows:

Corollary 3.2.54. *Let $\mathcal{T}$ be a TBox, Φ a set of atomic concepts, $n \in \mathbb{N}$ and D a q-concept containing only atomic concepts occurring in $\mathcal{T}$ but not in Φ and $n \geq 2^{\mathsf{size}(D)+\mathsf{size}(\mathcal{T})}$. Then D is satisfiable w.r.t. $\mathcal{T}$ iff D is satisfiable w.r.t. $\mathsf{UI}(linkless(C_{\mathcal{T}}^{(n)}), \Phi)$.*

$\mathsf{UI}(linkless(C_{\mathcal{T}}^{(n)}), \Phi)$ is called the *n-th approximation of a Φ-interpolant of $\mathcal{T}$* and as stated in the previous section on the UI-function, it can be computed in time linear in the size of the linkless normal form of $C_{\mathcal{T}}^{(n)}$. When checking the satisfiability of q-concepts D w.r.t. a Φ-interpolant of $\mathcal{T}$, it is important to know the maximal size of the q-concepts we want to test the satisfiability w.r.t. the Φ-interpolant. This size is then used to determine n as suggested in Wang et al. (2009) and then the n-th approximation of the Φ-interpolant can be used instead of the Φ-interpolant itself to check if the q-concept is satisfiable w.r.t. the Φ-interpolant of the TBox.

3.2.3.1 Implementation

Based on the precompilation of TBoxes into linkless normal form presented in Section 3.2.3, a prototypical knowledge compilation system was developed as a minor thesis by Günther (2009)[1] (see also (Furbach et al., 2009)). In order to increase performance by allowing the reuse of precompiled subconcepts, the implementation uses a graph structure to represent the result of the precompilation. The graph representation of a concept corresponds to the linkless normal form of the concept but allows to reuse subconcepts and is therefore convenient for implementation purposes. Please note that parts of this Section were already published in (Furbach et al., 2009).

In the following, we present the basic idea of this graph structure.

Definition 3.2.55 (R-Reachability Relations on Concepts)**.** *Let Σ be a description logic signature and C, $C' \in C^{\Sigma}$ be top level linkless concepts.*

- $\overset{\mathrm{R}}{\hookrightarrow}_{\mathrm{p}} \subseteq C^{\Sigma} \times C^{\Sigma}$ *is a relation defined as follows:*

  $C \overset{\mathrm{R}}{\hookrightarrow}_{\mathrm{p}} C'$ *iff there is a path P in C with the nonempty set of all universal role restrictions occurring in $P, \{\forall R.B_1, \ldots, \forall R.B_n\} \subseteq P$, and $C' \equiv B_1 \sqcap \ldots \sqcap B_n$.*

[1] Implementation available at `http://userp.uni-koblenz.de/~obermaie/LinklessNormalform.zip` [2015, December 17]

- $\overset{R}{\hookrightarrow} \subseteq C^\Sigma \times C^\Sigma$ *is a relation defined as follows:*

  $C \overset{R}{\hookrightarrow} C'$ *iff there is a path* P *in* C *with* $\exists R.A \in P$ *and the possibly empty set of all universal role restrictions* $\{\forall R.B_1, \ldots, \forall R.B_n\} \subseteq P$ *and* $C' \equiv A \sqcap B_1 \sqcap \ldots \sqcap B_n$.

If $C \overset{R}{\hookrightarrow} C'$, C' *is called* R-reachable *from* C. *If* $C \overset{\mathrm{R}}{\hookrightarrow}_{\mathrm{p}} C'$, C' *is called* potentially R-reachable *from* C. *If* $P \in \mathsf{paths}(C)$ *is the path mentioned in the definition of* $\overset{\mathrm{R}}{\hookrightarrow}_{\mathrm{p}}$ *or* $\overset{R}{\hookrightarrow}$ *respectively then* P *is called* path used to R-reach C' *from* C *or* path used to potentially R-reach C' *from* C *respectively. A concept* C' *is called* reachable *from concept* C, *denoted by* $C \hookrightarrow C'$, *if it is* R*-reachable from* C *for some role* R. *A concept* C' *is called* potentially reachable *from concept* C, *denoted by* $C \hookrightarrow_{\mathrm{p}} C'$, *if it is potentially* R*-reachable from* C *for some role* R. *Furthermore,* $\hookrightarrow^*$ *is the transitive reflexive closure of the union of all* $\overset{R}{\hookrightarrow}$ *relations and* $\hookrightarrow^*_{\mathrm{p}}$ *is the transitive reflexive closure of the union of all* $\overset{\mathrm{R}}{\hookrightarrow}_{\mathrm{p}}$ *relations.*

Example 3.2.56. *Consider the concept*

$$C = \exists R.A \sqcap (\forall R.(\neg A \sqcup B) \sqcup \neg A) \sqcup \exists R'.D.$$

C *has three different paths:*

$$P_1 = \{\exists R.A, \forall R.(\neg A \sqcup B)\}$$
$$P_2 = \{\exists R.A, \neg A\}$$
$$P_3 = \{\exists R'.D, \neg A\}$$

We list the different reachabilities:

$$C \overset{\mathrm{R}}{\hookrightarrow}_{\mathrm{p}} (\neg A \sqcup B),$$
$$C \overset{R}{\hookrightarrow} (A \sqcap (\neg A \sqcup B)),$$
$$C \overset{R}{\hookrightarrow} A,$$
$$C \overset{R'}{\hookrightarrow} D,$$
$$C \hookrightarrow D, (A \sqcap (\neg A \sqcup B)) \text{ and } A.$$

Note that it is possible that a concept is reachable or potentially reachable from another concept via several paths.

Since the removal of links preserves equivalence, for a concept C' which is R-reachable from C, both C' and $\mathsf{fulldissolventDL}(C')$ are called R-reachable from C.

Example 3.2.57. *Reconsider concept* C *given in Example 3.2.56:*

$$C = \exists R.A \sqcap (\forall R.(\neg A \sqcup B) \sqcup \neg A) \sqcup \exists R', D.$$

Concept $A \sqcap (\neg A \sqcup B)$ *is* R*-reachable from* C. $\mathsf{fulldissolventDL}(A \sqcap (\neg A \sqcup B) = B$ *is* R*-reachable from* C *as well.*

As mentioned afore, a path P is unsatisfiable iff the conjunction of its elements is unsatisfiable. Recall that, according to Lemma 3.2.3, there are four different possibilities for a path to be unsatisfiable:

1. $\bot \in P$.
2. there is $L \in P$ with $\overline{L} \in P$ for L a literal concepts.
3. $\exists R.C \in P$ with unsatisfiable concept C.
4. $\{\exists R.C, \forall R.D_1, \ldots, \forall R.D_n\} \subseteq P$ with unsatisfiable concept $C \sqcap D_1 \sqcap \ldots \sqcap D_n$.

3. and 4. can be restated as: There is an unsatisfiable concept which is reachable from C using P.

The basic idea of the graph representation of the linkless version of some concept C is to first compute the full dissolvent of C. In the second step, the full dissolvent of all concepts C' with $C \hookrightarrow^* C'$ or $C \hookrightarrow^*_{\mathrm{p}} C'$ is computed. This precompilation of concepts C' with $C \hookrightarrow^*_{\mathrm{p}} C'$ is necessary when we want to check subsumptions after the precompilation. This aspect is illustrated in the next example.

Example 3.2.58. *Consider the concept*

$$C = \forall R.\neg D \sqcap \forall R.\neg E.$$

There is no concept which is reachable from C since C does not contain any existential role restrictions. Checking a subsumption like

$$\forall R.\neg D \sqcap \forall R.\neg E \sqsubseteq \forall R.(\neg D \sqcap \neg E)$$

corresponds to checking the satisfiability of

$$C' = \forall R.\neg D \sqcap \forall R.\neg E \sqcap \exists R.(D \sqcup E).$$

There is a concept, which is reachable form C': $C' \overset{R}{\hookrightarrow} \big((D \sqcup E) \sqcap \neg D \sqcap \neg E\big)$.

Since the concepts which are reachable or potentially reachable from a concept via a path P only depend on the role restrictions occurring in P, we regard all paths containing the same set of role restrictions as equivalent.

Next we describe how to represent a concept C as a rooted directed graph i.e. a directed graph with exactly one source. The graph consists of two different types of nodes: A set of *concept nodes* CN containing top level linkless concepts and a set of *path nodes* PN containing paths of these concepts. The source node is a concept node consisting of the top level linkless version of C. The set of edges is $E \subseteq (CN \times PN) \cup (PN \times CN)$, meaning that concept nodes and path nodes occur alternating in all paths in the graph. A concept node C has a successor node for each set of equivalent paths in C. Furthermore, there is an edge from a path node to the concept nodes of reachable and potentially reachable concepts. Each edge connecting a path node with a concept node is labeled with the set of role restrictions used to reach or potentially reach the respective concept. Each path

node has exactly one preceding concept node, whereas a concept node can have more than one preceding path node.

In the following definition, for a path P,

$$\mathsf{role}(P) = \{QR.C \in P \mid Q \in \{\exists, \forall\}, \text{ for some concept } C \text{ and some role } R\}$$

denotes the set of role restrictions occurring in P. Furthermore, we use $\rightsquigarrow$ to denote the union of the reaches and potentially reaches relations:

$$\rightsquigarrow = \hookrightarrow \cup \hookrightarrow_{\mathrm{p}}$$

Intuitively $C \rightsquigarrow C'$ holds if $C \hookrightarrow C'$ or $C \hookrightarrow_{\mathrm{p}} C'$.

Definition 3.2.59 (Linkless Graph)**.** *Let C be concept. The precompilation of a concept C is a rooted directed graph (N, E), called* linkless graph*, where*

1. *The root node is* $\mathsf{fulldissolventDL}(C)$.
2. $N = CN \cup PN$.
   a) *CN is the set of concept nodes defined as:*

$$\begin{aligned} CN = &\{\mathsf{fulldissolventDL}(C)\} \cup \\ &\{C' \mid C \rightsquigarrow^* C' \text{ and } C' \text{ a top level linkless concept.}\} \end{aligned}$$

   b) *PN is the set of path nodes defined as:*

$$\begin{aligned} PN = \{P \mid \exists C' \in CN \text{ such that } P \subseteq \mathsf{paths}(C') \text{ and} \\ \forall P', P'' \in P \ \mathsf{role}(P') = \mathsf{role}(P'') \text{ holds.}\} \end{aligned}$$

3. *The set of edges E is defined as:*

$$\begin{aligned} E = &\{(C', P) \mid C' \in CN \text{ and } P \in PN \text{ and } P \subseteq \mathsf{paths}(C)\} \cup \\ &\{(P, C') \mid C' \in CN \text{ and } P \in PN \text{ and } \exists C'' \in CN \text{ with } C'' \rightsquigarrow C' \text{ and} \\ &\qquad P' \text{ is a path used to reach or potentially reach } C' \text{ from } C'', \\ &\qquad \text{for some path } P'.\} \end{aligned}$$

   Each edge of the form (P, C') with $C' \in CN$ and $P \in PN$ is labeled with $\mathsf{role}(P')$ for some $P' \in P$.

Furthermore, for all $P \in PN$ holds: $|\{C' \mid \langle C', P\rangle \in E\}| = 1$.

Example 3.2.60. *Consider for the linkless concept:*

$$C = (B \sqcap \neg E) \sqcup \Big(\big(B \sqcup \neg A \sqcup (\exists R.A \sqcap A)\big) \sqcap \exists R.E \sqcap \forall R.\neg A\Big)$$

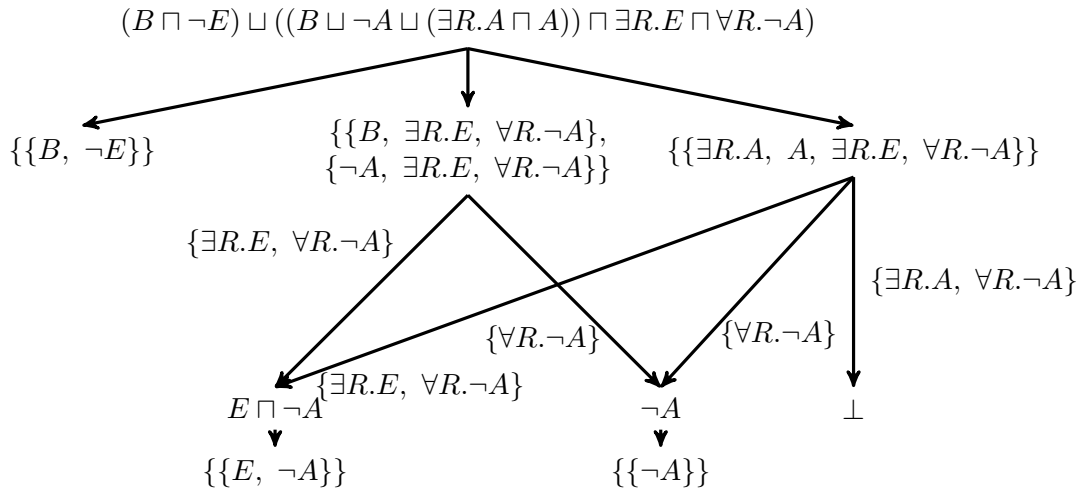


Figure 3.3: Linkless Graph for $(B \sqcap \neg E) \sqcup \Big(\big(B \sqcup \neg A \sqcup (\exists R.A \sqcap A)\big) \sqcap \exists R.E \sqcap \forall R.\neg A\Big)$.

Its linkless graph with root C is depicted in Figure 3.3. C has the four paths

$$P_1 = \{B, \neg E\},$$
$$P_2 = \{B, \exists R.E, \forall R.\neg A\},$$
$$P_3 = \{\neg A, \exists R.E, \forall R.\neg A\},$$
$$P_4 = \{\exists R.A, A, \exists R.E, \forall R.\neg A\}.$$

As mentioned afore, we regard paths containing the same set of role restrictions as equivalent. There are three sets of equivalent paths: $\{P_1\}$, $\{P_2, P_3\}$ and $\{P_4\}$. For each set of equivalent paths, there is a successor path node. In the next step, reachable concepts are considered: For instance the concept $E \sqcap \neg A$ is reachable via the paths $\{P_2, P_3\}$ using the role restrictions $\{\exists R.E, \forall R.\neg A\}$. Therefore there is an edge from the second path node to the concept node $E \sqcap \neg A$ labeled $\{\exists R.E, \forall R.\neg A\}$. In the same way, the precompilation of all (potentially) reachable concepts are combined with the path nodes.

When answering queries with respect to a TBox it is necessary to restrict reasoning such that only models of the TBox are considered. As described by Franz Baader and Diego Calvanese and Deborah L. McGuinness and Daniele Nardi and Peter F. Patel-Schneider (2003) and introduced in Section 3.2.3 a TBox

$$\mathcal{T} = \{C_1 \sqsubseteq D_1, \ldots, C_n \sqsubseteq D_n\}$$

can be represented as a metaconstraint

$$C_{\mathcal{T}} = (\neg C_1 \sqcup D_1) \sqcap \ldots \sqcap (\neg C_n \sqcup D_n).$$

The idea of the linkless graph can be directly extended to represent precompiled TBoxes. This can be done by constructing the linkless graph for $C_{\mathcal{T}}$, but instead of just considering

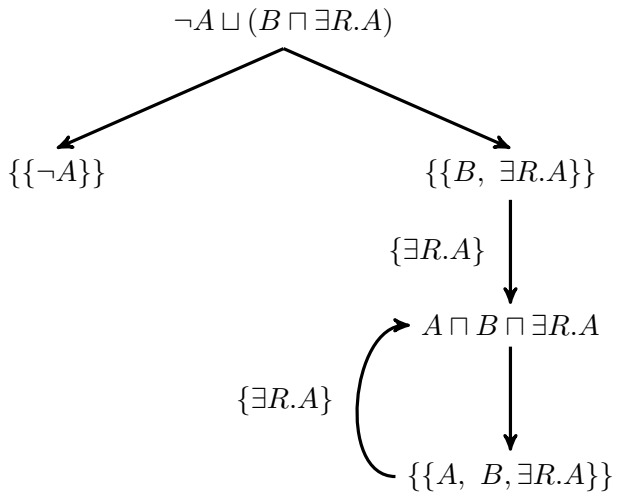


Figure 3.4: Linkless Graph for the TBox $\mathcal{T} = \{A \sqsubseteq B \sqcap \exists R.A\}$.

the concept nodes, each concept node C must also fulfill $C_\mathcal{T}$ meaning that we replace each concept node C by $\mathsf{fulldissolventDL}(C \sqcap C_\mathcal{T})$.

Note that the linkless graph for a TBox typically contains cycles.

Example 3.2.61. *Consider the TBox*

$$\mathcal{T} = \{A \sqsubseteq B \sqcap \exists R.A\}$$

with the corresponding metaconstraint

$$C_\mathcal{T} = \neg A \sqcup (B \sqcap \exists R.A).$$

Figure 3.4 depicts the linkless graph for $\mathcal{T}$. There are two different paths in $C_\mathcal{T}$, which constitute the two subsequent path nodes. The path node $\{\{B, \exists R.A\}\}$ can be used to reach the concept $A \sqcap C_\mathcal{T}$ which is equivalent to the top level linkless concept $A \sqcap B \sqcap \exists R.A$, which therefore labels this concept node. Since $A \sqcap B \sqcap \exists R.A$ has the path $\{\{A, B, \exists R.A\}\}$, the graph contains a cycle.

In the worst case, transforming a concept into linkless normal form causes an exponential blowup. Since the linkless graph for a concept is just a syntactical variant to the linkless normal form for this concept, it is not surprising that transforming a concept into a linkless graph can cause the same blowup. Namely transforming the concept nodes into top level linkless concepts can generate an exponential blowup. Furthermore, for a concept C there can be exponentially many concepts C' with $C \rightsquigarrow^* C'$ in the worst case. If a concept C contains r different roles each with n existential role restrictions and m universal role restrictions, the number of concepts C' with $C \hookrightarrow^* C'$ is $r \cdot n \cdot 2^m$ in the worst case. We call these concepts C' concepts which are reachable* from C. Furthermore, the number of concepts C' with $C \hookrightarrow^*_\mathrm{p} C'$ is $r \cdot 2^m$ in the worst case. We call these

concepts C' concepts which are potentially reachable* from C. The 2^m in both cases arises from the fact that a set of m universal role restrictions has 2^m different subsets. And each of these subsets corresponds to one potentially reachable concept. However in real world ontologies the number of reachable concepts is smaller. Furthermore precompiling a TBox never increases the number of reachable concepts, contrariwise it usually decreases it, because some of the reachable concepts turn out to be unsatisfiable, are simplified to $\bot$ and are therefore summarized. For example for the amino-acid[2] ontology $r = 5$, $m = 3$ and $n = 5$. So in the worst case, there are 200 reachable* concepts. But in reality, before the precompilation there are 170 and after the precompilation 154 reachable concepts.

Our implementation first flattens the TBox (Rudolph et al., 2008) i.e. all nested role restrictions are removed. This is similar to the flattening of a concept presented in Definition 3.2.25. Flattening is done by replacing each occurrence of $\exists R.C$ ($\forall R.C$) in an axiom by $\exists R.C'$ ($\forall R.C'$) with C' a fresh atomic concept. Furthermore the axioms $C' \sqsubseteq C$ and $C \sqsubseteq C'$ are added. This transformation is repeated recursively until no nested role restrictions are left and for all role restriction $\exists R.B$ or $\forall R.B$, B is a fresh atomic concept. Removing nested role restrictions ensures that only concepts of the form $D \sqcap \mathsf{fulldissolventDL}(C_{\mathcal{T}})$ are reachable or potentially reachable, where D is a conjunction of fresh atomic concepts and further that every concept which is reachable in more than one step is also reachable in exactly one step.

Different ontologies from the literature (Foodswap[3], Ribosome[4], Nautilus-exceptions[5], Koala[6] and Amino-acid were used to test the implementation. We ignored features not belonging to $\mathcal{ALC}$ occurring in these ontologies. Since the removal of links from a TBox in the worst case produces an exponential blowup, it is crucial to find out if this blowup occurs when precompiling real ontologies. Figure 3.2 gives information on that and shows that 11 links were removed from the Foodswap ontology, which caused the ontology only to grow from size 130 to 159. Furthermore, 19 links were removed from the Ribosome ontology which caused the TBox size only to reduplicate. Figure 3.2 shows that for none of the precompiled ontologies an exponential blowup occurred. Another point which is interesting is the number of different reachable and potentially reachable concepts depending on the number of different roles occurring in an ontology. Figure 3.3 shows that for the ontologies we considered, the number of reachable and potentially reachable worlds is manageable. Since the performance of query answering depends on the size of the linkless graph, the experiments confirm the fact that the precompilation of a TBox into a linkless graph is worthwhile.

As mentioned before, the linkless graph for a concept or a TBox is just a syntactical variant of the linkless normal form for the concept or TBoxe introduced in the previous sections. Therefore these experimental results can be seen as experimental results for

[2] `https://bioportal.bioontology.org/ontologies/AMINO-ACID`, [2015, December 17]
[3] `http://www.mindswap.org/dav/ontologies/commonsense/food/foodswap.owl`, [2015, December 17]
[4] `http://130.88.198.11/co-ode-files/ontologies/Ribosome.owl`, [2015, December 17]
[5] `http://130.88.198.11/co-ode-files/ontologies/Nautilus-exceptions.owl`, [2015, December 17]
[6] Available at `http://protegewiki.stanford.edu/wiki/Protege_Ontology_Library`, [2015, December 17]

Ontology	no. of roles	Size of flat TBox	Size of linkless TBox	removed links
Foodswap	1	130	159	11
Ribosome	1	133	354	19
Nautilus-ex.	2	208	717	40
Koala	5	272	362	19
Amino-acid	5	1215	11024	130

Table 3.2: Result of Precompiling Different Ontologies.

Ontology	reachable concepts	pot. reachable concepts
Foodswap	0	17
Ribosome	32	7
Nautilus-ex.	9	3
Koala	18	9
Amino-acid	154	88

Table 3.3: Result of Precompiling Different Ontologies: Reachable and Potentially Reachable Concepts.

the linkless normal form as well. Furthermore, it is possible to use the linkless graph to answer queries. Currently the implementation allows satisfiability and subsumption checking. See (Furbach et al., 2009) for details.

3.2.4 Related Work

In this section work related to knowledge compilation in description logics is considered. In Section 3.2.4.1, we start with an introduction of prime implicate normal form for $\mathcal{ALC}$ concepts, which Bienvenu (2008a) suggests as a target language for knowledge compilation purposes. Section 3.2.4.2 contains a brief introduction of some other normal forms for description logics. Other related preprocessing techniques used for reasoning in description logics are presented in Section 3.2.4.3.

3.2.4.1 Prime Implicate Normal Form for $\mathcal{ALC}$ Concepts

Bienvenu (2008a) introduces a normal form called *prime implicate normal form* for $\mathcal{ALC}$ concepts. As the name suggests, this normal form is closely related to the prime implicate normal form known for propositional logic which we recalled in Section 3.1.2.3. Unless stated otherwise, all definitions given in this section are taken from Bienvenu (2008a) and Bienvenu (2008b) and are adapted to fit to our notation. In the remainder of this section, we are only considering concepts given in the description logic $\mathcal{ALC}$ and therefore the term concept actually means $\mathcal{ALC}$ concepts. In order to present the definition of

prime implicants of concepts, it is necessary to introduce the notion of clausal and cubal concepts.

Definition 3.2.62 (Extended Literal/Clausal/Cubal Concepts (Bienvenu, 2008a))**.** *Let* $\Sigma = (N_C, N_R, N_I)$ *be a description logic signature.* Extended literal concepts L, clausal concepts Cl *and* cubal concepts Cb *are formed according the following syntax rule:*

$$\begin{aligned} L &\rightarrow \top \mid \bot \mid A \mid \neg A \mid \forall R.D \mid \exists R.D \\ Cl &\rightarrow L \mid Cl \sqcup Cl \\ Cb &\rightarrow L \mid Cb \sqcap Cb \\ D &\rightarrow \top \mid \bot \mid A \mid \neg A \mid D \sqcap D \mid D \sqcup D \mid \forall R.D \mid \exists R.D \end{aligned}$$

where $A \in N_C$ *and* $R \in N_R$.

Please note that Bienvenu (2008b) introduces the afore defined *extended literal concept* as *literal concept*. However we already use the term literal concept with a different meaning. This is why we deviate form the nomenclature in Bienvenu (2008b).

Using the above introduced notion of a clausal concept, prime implicates of a concept can be defined like in propositional logic.

Definition 3.2.63 (Prime Implicate (Bienvenu, 2008a))**.** *Let* C *be a concept and* Cl *be a clausal concept.* Cl *is an* implicat *of* C *iff* $\models C \sqsubseteq Cl$. *A clausal concept* Cl *is a* prime implicate *of* C *iff:*

1. Cl *is an implicate of* C*, and*
2. *if* Cl' *is an implicate of* C *such that* $\models Cl' \sqsubseteq Cl$ *then* $\models Cl \sqsubseteq Cl'$.

A prime implicate is an implicate which does not subsume any other implicates. This corresponds to the notion of prime implicates in propositional logic given in Definition 3.1.38.

For a clausal concept Cl, $\exists R(Cl)$ is the set of concepts C such that $\exists R.C$ is a disjunct of Cl. $\forall R(Cl)$ is used respectively.

Example 3.2.64. *Consider the clausal concept*

$$Cl = A \sqcup \neg B \sqcup \exists R.(E \sqcap \forall R.C) \sqcup \forall R.E \sqcup \exists R.A$$

Then $\exists R(Cl)$ *and* $\forall R(Cl)$ *are given as follows:*

$$\begin{aligned} \exists R(Cl) &= \{E \sqcap \forall R.C,\ A\} \\ \forall R(Cl) &= \{E\} \end{aligned}$$

According to Definition 3.1.38, the prime implicate normal form of a propositional logic formula is defined as the conjunction of its prime implicates. The naive way to define a prime implicate normal form for $\mathcal{ALC}$ concepts would be to do the same by defining it as the conjunction of its prime implicates as follows.

Definition 3.2.65 (Naive Prime Implicate Normal Form)**.** *A concept C is in* naive prime implicate normal form *if it is the conjunction of its prime implicates.*

However the resulting normal form does not have the same nice properties like the corresponding normal form for propositional logic. In propositional logic prime implicate normal form allows to efficiently perform subsumption testing, since subsumption testing can be reduced to consistency checking. Bienvenu (2008b) introduces an example illustrating that the naive definition of prime implicate normal form does not allow an efficient subsumption test: Consider the concepts $\exists R.C_1$ and $\exists R.C_2$ with C_1 and C_2 in NNF. Both $\exists R.C_1$ and $\exists R.C_2$ are their own prime implicates and therefore would be in prime implicate normal form according to the naive definition. For arbitrary NNF concepts C_1 and C_2, C_1 subsumes C_2 only if $\exists R.C_1$ subsumes $\exists R.C_2$. As argued above, both $\exists R.C_1$ and $\exists R.C_2$ are in prime implicate normal form according to the naive definition. This means that subsumption checking between arbitrary NNF concepts can be reduced to subsumption checking between concepts in prime implicate normal form. Since subsumption checking for arbitrary NNF concepts is known to be PSPACE-complete (Schild, 1991), this implies that subsumption checking has to be PSPACE-complete.

In order to allow an efficient subsumption check for concepts in prime implicate normal form, it is necessary to define the notion of prime implicate normal form for $\mathcal{ALC}$ concepts differently.

Definition 3.2.66 (Prime Implicate Normal Form (Bienvenu, 2008a))**.** *A concept C is in* prime implicate normal form *iff it satisfies one of the following conditions:*

1. $C = \bot$.
2. $C = \top$.
3. $C \neq \bot$ *and* $C \neq \top$ *and* $C = Cl_1 \sqcap \ldots \sqcap Cl_p$ *where*
   - 3.a) $\not\models Cl_i \sqsubseteq Cl_j$ *for* $i \neq j$,
   - 3.b) *each prime implicate of C is equivalent to some conjunct Cl_i,*
   - 3.c) *every Cl_i is a prime implicate of C such that*
     - i. *if D is a disjunct of Cl_i then* $\not\models Cl_i \sqsubseteq (Cl_i \setminus D)$ *or* $\not\models (Cl_i \setminus D) \sqsubseteq Cl_i$,
     - ii. $|\exists R(Cl_i)| \leq$ *for every role R,*
     - iii. *if* $E \in \exists R(Cl_i) \cup \forall R(Cl_i)$ *for some R, then E is in prime implicate normal form,*
     - iv. *if* $E \in \exists R(Cl_i)$ *and* $F \in \forall R(Cl_i)$ *then* $\models E \sqsubseteq F$.

Since the purpose of the different parts of this definition are not immediately obvious, we will explain the different parts like it is done in Bienvenu (2008b).

1. and 2. make sure that unsatisfiable concepts are represented as $\bot$ and tautological concepts as $\top$. Due to this, unsatisfiability and validity of a concept in prime implicate normal form can be tested in constant time.

2. considers satisfiable concepts which are not valid. These concepts have to be presented as a conjunction of prime implicates.
   a) specifies the fact that the conjuncts are prime implicates.
   b) specifies that every prime implicate is part of the conjunction.
   c) i. specifies that the prime implicates do not contain unnecessary concepts.
      ii. states that each conjunct is allowed to contain at most one existential role restriction per role.
      iii. specifies that concepts occurring in the scope of a role restriction are required to be in prime implicate normal form. Let us reconsider the concepts $\exists R.C_1$ and $\exists R.C_2$ (with C_1 and C_2 being arbitrary NNF concepts) mentioned directly before Definition 3.2.66. According to the naive definition of prime implicate normal form given in Definition 3.2.65, these two concepts are in naive prime implicate normal form. However when using Definition 3.2.66, these concepts are not in prime implicate normal form since the concepts C_1 and C_2 are not in prime implicate normal form. The fact that concepts in the scope of role restrictions have to be transformed into the normal form as well is not really astonishing. Intuitively it is necessary to have those concepts in prime implicate normal form as well in order to ensure efficient reasoning.
      iv. demands that if a prime implicate contains disjuncts $\exists R.E$ and $\forall R.F$ then $\models E \sqsubseteq F$. According to Bienvenu (2008b), this is a very important property because it allows to treat universal role restrictions separately form existential role restrictions during subsumption checking. This property ensures that whenever a universal role restriction is subsumed by a clausal concept, it is subsumed by some universal role restriction in the clausal concept as well.

We will not go into the details how to compute the set of prime implicates for a concept and refer the reader to (Bienvenu, 2008b). However it is interesting to note that, according to Bienvenu (2008b) the number of non-equivalent prime implicates of a given concept C with n different literal concepts occurring in C is bounded from above by n^{2^n}. Furthermore, transforming a concept C into prime implicate normal form can lead to an doubly-exponential blowup. See (Bienvenu, 2008b) for a proof.

Properties of Prime Implicate Normal Form for $\mathcal{ALC}$ Concepts Next, we introduce different queries which can be performed in polynomial time on concepts in prime implicate normal form. As already mentioned above, it follows from Definition 3.2.66 that concepts given in prime implicate normal form can be tested for consistency and validity in constant time. Furthermore, given two concepts C_1 and C_2 in prime implicate normal form, it can be efficiently tested if C_1 is subsumed by C_2. This subsumption test can be performed in time linear in $\mathsf{size}(C_1) + \mathsf{size}(C_2)$ (meaning in at most time quadratic to $\mathsf{size}(C_1 + C_2)$).

In addition to queries which can be performed efficiently for concepts in prime implicate normal form it is interesting to investigate which transformations on concepts in this normal form can be performed in polynomial time. One of the transformations interesting in the matters of knowledge compilation in propositional logic listed in Darwiche and Marquis (2002) is projection. As stated by Definition 3.1.4, projecting a propositional logic formula $F \in F^V_{prop}$ on a set of variables A leads to a formula F' with signature A which has the same consequences as F w.r.t. formulae constructed from A variables. Projection is dual to the notion of forgetting. Projecting a formula $F \in F^V_{prop}$ on a set of variables A leads to the same result as forgetting the variables in $V \setminus A$ from F. In description logic, forgetting corresponds to uniform interpolation as given in Definition 3.2.39 and is undoubtedly a very interesting transformation. For concepts C given in prime implicate normal form and a set of concept symbols Φ, it is possible to construct the Φ-interpolant in time linear in the size of C. See (Bienvenu, 2008b) for a proof of this property and an algorithm performing the computation of Φ-interpolants from concepts in prime implicate normal form.

Comparison of Prime Implicate Normal form and Linkless Normal Form Comparing the prime implicate normal form to the linkless normal form for concepts reveals that both normal forms allow constant time consistency check. In contrast to the linkless normal form, the prime implicate normal form allows a constant time validity check as well. Both normal forms allow efficient subsumption checking. However in the case of the linkless normal form, these subsumption checks are restricted: For a linkless concept C, a subsumption $C \sqsubseteq D$ can only be checked efficiently if D is a q-concept. Both normal forms allow linear projection. It is noteworthy that compilation of a concept into prime implicate normal form can cause a doubly-exponential blowup as opposed to the single-exponential blowup which can be caused when compiling a concept into linkless normal form. Furthermore, until now there is no extension of the prime implicate normal form for TBoxes.

3.2.4.2 Other Normal Forms

One of the early approaches to knowledge compilation with regards to description logics, is presented by Selman and Kautz (1996), where $\mathcal{FL}$ concepts are approximated by $\mathcal{FL}^-$concepts. $\mathcal{FL}$ concepts C, D can be constructed according to the following syntax rule

$$C, D \quad \rightarrow \quad A \mid C \sqcap D \mid \forall R.C \mid \exists R$$

where A is an atomic concept and R a so called *role restriction*. Please note that the notion of role restrictions in $\mathcal{FL}$ differs from the notion of role restrictions used in the thesis at hand. In $\mathcal{FL}$, a role restriction can be used to construct a subrole $R_{|C}$ of a role R. These role restrictions have the following semantics:

$$(R_{|C})^{\mathcal{I}} = \{(x, y) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid (x, y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$$

Semantics of $\mathcal{FL}$ concepts is given as defined in Definition 2.4.8 together with

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

The description logic $\mathcal{FL}^-$ is obtained from $\mathcal{FL}$ be omitting role restrictions. Subsumption checking in $\mathcal{FL}$ is coNP-hard (Franz Baader and Diego Calvanese and Deborah L. McGuinness and Daniele Nardi and Peter F. Patel-Schneider, 2003). In contrast to this, subsumption checking in $\mathcal{FL}^-$ is in P and can be done by structural comparison between concepts. The idea introduced by Selman and Kautz (1996) is to approximate an $\mathcal{FL}$ concept C by two $\mathcal{FL}^-$ concepts, namely the concept C' which is the most general more specific $\mathcal{FL}^-$ concept and concept C'' which is the most specific more general $\mathcal{FL}^-$ concept.

For the multi modal logic $\mathcal{K}_n$ Kracht (1999) introduces a normal form called *standard formulae of degree d* which is related to the linkless normal form introduced in Section 3.2.2. Recall that a concept given in linkless normal form is supposed to be in propagated $\exists$-NF and that the propagated $\exists$-NF includes the $\forall$-NF. The d in standard formulae of degree d describes the depth of modal logic formula and is defined analogous to the depth function given in Definition 2.4.6. Meaning that a standard formula of degree 0 does not contain any modal operators and of degree 1 contains modal operators but no nested modal operators. Standard formulae of degree d impose restrictions on the occurrence of $\Box_i$ operators claiming that there are no conjunctively combined $\Box_i C$ and $\Box_i D$. Since $\mathcal{K}_n$ and the description logic $\mathcal{ALC}$ are closely related, this corresponds to the claim that a concept in $\forall$-NF does not allow occurrences of conjunctively combined $\forall R.C$ and $\forall R.D$. Furthermore, standard formulae of degree d are not allowed to contain $\Diamond_i C$ which is conjunctively combined with both $\Box_i B_1$ and $\Box_i B_2$. This corresponds to the property of $\exists$-NF concepts which are not allowed to contain $\exists R.C$ which is conjunctively combined with both $\forall R.B_1$ and $\forall R.B_2$. A concept given in linkless normal form is in propagated $\exists$-NF as well, meaning that for all subconcepts $\exists R.C$ occurring conjunctively combined with $\forall R.D$, $C \equiv C \sqcap D$. This property is crucial for the properties of linkless concepts. Standard formulae of degree d do not claim a corresponding property: For conjunctively combined $\Diamond_i C$ and $\Box_i D$ no relation between C and D is demanded. Another difference between linkless concepts and standard formulae of degree d is the structure of the overall formula. Linkless normal form only claims the concept to be in NNF, thus allowing rather succinct concepts like

$$(C \sqcup (A \sqcap B)) \sqcap (E \sqcup F \sqcup G)$$

whereas the corresponding $\mathcal{K}_n$ formula

$$(c \vee (a \wedge b)) \wedge (e \vee f \vee g)$$

is not a standard formula of degree 0. The corresponding standard formula of degree 0 is much longer, since it is supposed to be a disjunction of conjunctions of variables or negated variables:

$$(a \wedge b \wedge e) \vee (a \wedge b \wedge f) \vee (a \wedge b \wedge g) \vee (c \wedge e) \vee (c \wedge f) \vee (c \wedge g).$$

Furthermore, since standard formulae of degree d are allowed to contain conjunctively combined a and $\neg a$ for variables a, standard formulae of degree d do not share the nice properties of linkless concepts.

Another normal form for description logics which is supposed to be suitable for knowledge compilation purposes is introduced in Zou, Liu, and Lv (2012); Liu, Gu, Zou, Huang, and Li (2013). This approach does not clearly state what kind of description logic they are able to handle, thus we cannot compare our approach to their method.

3.2.4.3 Structural Subsumption Checking, Normalization and Absorption

There are several techniques for description logics which are related to knowledge compilation techniques. Structural subsumption algorithms (Franz Baader and Diego Calvanese and Deborah L. McGuinness and Daniele Nardi and Peter F. Patel-Schneider, 2003) are used to perform subsumption checks on two concepts. For this, both concepts are transformed into a normal form and afterwards the structure of these normal forms is compared. However these algorithms typically have problems with more expressive description logics. Especially general negation, which is an important feature in the application of description logics, is a problem for those algorithms. In contrast to structural subsumption algorithms, the linkless normal form is able to handle general negation without problems.

Normalization (Balsiger and Heuerding, 1998) is a preprocessing technique for description logics, which eliminates redundant operators in order to determine contradictory as well as tautological parts of a concept. In many cases this technique is able to simplify subsumption and satisfiability problems.

Another preprocessing technique frequently used is absorption (Franz Baader and Diego Calvanese and Deborah L. McGuinness and Daniele Nardi and Peter F. Patel-Schneider, 2003). General inclusion axioms $C \sqsubseteq D$ constitute a major source of complexity during reasoning as they add a disjunction $\neg C \sqcup D$ to each node in a tableau. This is why absorption tries to eliminate general inclusion axioms from a knowledge base. Given a general inclusion axiom $C \sqsubseteq D$ with C and D arbitrary concepts, absorption aims at creating $A \sqsubseteq D'$ with A an atomic concept which is called a *primitive definition*. This can be accomplished by using the following equivalences:

$$\begin{array}{lcl} C_1 \sqcap C_2 \sqsubseteq D & \iff & C_1 \sqsubseteq D \sqcup \neg C_2 \\ C \sqsubseteq D_1 \sqcap D_2 & \iff & C \sqsubseteq D_1 \text{ and } C \sqsubseteq D_2 \end{array}$$

This primitive definition $A \sqsubseteq D'$ can be merged into another primitive definition $A \sqsubseteq C'$ to $A \sqsubseteq C' \sqcap D'$ reducing the number of general inclusion axioms. Both absorption and normalization aim at increasing the performance of tableau based reasoning procedures. In both cases the preprocessing does not lead to a normal form and therefore these techniques cannot be directly compared to the linkless normal form.

3.2.4.4 View Materialization

The basic idea of knowledge compilation is related to the idea of view materialization in data bases (Blakeley, Larson, and Tompa, 1986). A materialized view corresponds to a database object which contains the result of a query. View materialization aims at speeding up query processing by keeping views which are frequently accessed materialized. The

main difference to knowledge compilation is the fact that knowledge compilation is query independent, meaning that the precompiled version of the knowledge base can be used to answer arbitrary queries efficiently. In contrast to that, in view materialization only frequently accessed views are materialized. As soon as a unexpected view is demanded, the materialization does not provide an efficiency benefit.

3.2.4.5 Reduction of $\mathcal{SHIQ}$ Knowledge Bases into Datalog Programs

Motik (2006) introduces an approach to reduce $\mathcal{SHIQ}$ knowledge bases into datalog programs such that the original knowledge base and the datalog program entail the same set of ground facts. Like in the case of knowledge compilation, the reduction into a datalog program can cause an exponential blowup: According to Motik (2006), the number of rules in the datalog program can be exponential in the size of the original knowledge base. Opposed to knowledge compilation, the main advantage of reducing knowledge bases to datalog programs is that it is possible to apply different optimization techniques like the magic sets transformation to the resulting datalog programs without any changes. Furthermore, if the datalog program resulting from the reduction is not disjunctive, queries can be answered efficiently.

3.3 Future Work

Now we introduce three different directions interesting for future work in the area of this chapter.

The basic idea of the first direction for future work is to adapt other target languages known from propositional logic to description logic by using the complete propagated $\exists$-NF as a basis. When checking the satisfiability of an arbitrary $\mathcal{ALC}$ concept C, it is possible that C contains conjunctively combined subconcepts of the form $\exists R.C_1$ and $\forall R.C_2$ such that C_1 and C_2 both are satisfiable, but $C_1 \sqcap C_2$ are unsatisfiable. Depending on the structure of C, this can cause concept C to be unsatisfiable. For concepts in complete propagated $\exists$-NF as introduced in Section 3.2.1, this cannot happen. The main idea of the complete propagated $\exists$-NF is to summarize all concepts occurring in the scope of a role restriction as far as possible. This ensures that whenever there is a subconcept of the form $\exists R.C_1$ or $\forall R.C_2$ in a concept in complete propagated $\exists$-NF, there is no other $\forall R.C_3$ which has an influence on C_1 or C_2. In other words, whenever a subconcept of the form $\exists R.C_1$ or $\forall R.C_2$ occurs, it is ensured that C_1 or C_2 already contains all information relevant for the R-successor. This is why the complete propagated $\exists$-NF provides a suitable basis for adapting target languages used for knowledge compilation purposes for propositional logic to $\mathcal{ALC}$. In this chapter, the linkless normal form well-known for propositional logic is adapted to both $\mathcal{ALC}$ concepts. To accomplish this, the concept first has to be transformed into complete propagated $\exists$-NF. Then links are removed from the topmost level of the concept as well as from all concepts C occurring in the scope of a role restriction. One interesting line of research would be to use the complete propagated $\exists$-NF to adapt other traget languages known from propositional logic like the DNNF to $\mathcal{ALC}$ concepts. This could be done by using the same procedure as for the transformation

into linkless normal form. It would be interesting to investigate the properties of the so created normal forms.

The second direction interesting for future work is be the use of a SAT solver for satisfiability checking of concepts. This idea is not completely new. Sebastiani and Vescovi (2009) suggest to encode description logic concepts into propositional logic formulae and then call a SAT solver for satisfiability checking. In contrast to this, we suggest to use the complete propagated $\exists$-NF as a basis for the satisfiability test. The rough idea of this is as follows: Due to the restrictions on the concepts occurring in the scope of a role restrictions, it is possible to test the satisfiability of a concept C by testing the satisfiability of $\mathsf{prop}(C)$ with the help of a SAT solver. If $\mathsf{prop}(C)$ is unsatisfiable, C is unsatisfiable as well. If $\mathsf{prop}(C)$ is satisfiable, there are two possibilities: 1. C is satisfiable or 2. C is unsatisfiable and contains a contradiction in the scope of an existential role restriction. Due to the structure of concepts in complete propagated $\exists$-NF it is not possible that C contains $\exists R.C_1$ and $\forall R.C_2$ such that C_1 and C_2 both are satisfiable, but $C_1 \sqcap C_2$ are unsatisfiable. Hence, if the SAT solver finds a model for $\mathsf{prop}(C)$, this model has to be examined. If this model does not assign *true* to any atom of the form $\exists rc$, this corresponds to case 1. and C is satisfiable. If the model assigns true to atoms of the form $\exists rd$ and D is the corresponding concept occurring in the corresponding existential role restriction, the SAT solver has to be called for $\mathsf{prop}(D)$ for each of these atoms. An unsatisfiable result for one of these calls implies that the constructed model is not possible and has to be revised.

Another interesting direction for future research is the question if the normal forms developed in this chapter can be extended to more expressive description logics. The basic idea of the linkless normal form is to remove inconsistencies from the concept. When extending this normal form to more expressive description logics, it is important to know how contradictory concepts can be constructed in the description logic under consideration. For example, extending $\mathcal{ALC}$ with qualified number restrictions leads to the description logic $\mathcal{ALCQ}$ and allows to formalize concepts like a man with at least two sons and at most one daughter as

$$Male \sqcap \geq 2 hasChild.Male \sqcap \leq 1 hasChild.Female.$$

With these qualified number restrictions one can easily construct contradictory concepts like for example the following one

$$Male \sqcap \geq 2 hasChild.Male \sqcap \leq 1 hasChild.Male.$$

which is a man with at least two sons and at most one son. These contradictions constructed using qualified number restrictions have to be taken into account when adapting the linkless normal form to $\mathcal{ALCQ}$.

4 Evolution of the Instance Level of Knowledge Bases

It is a well known fact that knowledge bases usually are not static. In practice, they are subject to frequent changes and even the construction of a knowledge base can be seen as an iterative process. It is not astonishing that this process is error prone and often results in inconsistent ontologies. Even for smaller ontologies it is not possible for humans to determine how to remove those inconsistencies manually. In this chapter, we focus on changes affecting the *instance level* of a knowledge base. By instance level the subset of a knowledge base consisting of ground unit clauses in case of first-order logic or the set of unit clauses in case of propositional logic is understood. When considering description logic knowledge bases, the instance level denotes the ABox. When speaking about changing a knowledge base, it is traditionally distinguished between *update* and *revision*. For updates, the knowledge base is assumed to describe a certain state of the world and changes mean that newer information concerning the present is introduced into the knowledge base. In contrast to this, in the context of revision, the world is considered to be static and by revision new information on the world is incorporated assuming that older knowledge is less reliable than the introduced new information. The methods introduced in this chapter all implement revision. Throughout this chapter, *evolution* is used as a collective term for several operations revising the instance level of a knowledge base, namely insertion and deletion of instance assertions along with repairing an instance level which is inconsistent w.r.t. the rest of the knowledge base.

A well known work in this field is without doubt the so-called AGM theory (Alchourron, Gärdenfors, and Makinson, 1985), named after the names of its proponents, Alchourrón, Gärdenfors, and Makinson. This theory establishes the so-called AGM postulates stating properties a change operation has to fulfill in order to be considered rational. As mentioned before, in this chapter we will introduce operations for the evolution of the instance level of description logic knowledge bases. However it is not straightforward how to apply the AGM theory to description logics. One issue is the fact that the change operations considered in the AGM theory are applied to so-called *belief sets* which are deductively closed sets of formulae. These belief sets do not distinguish between knowledge explicitly stated and knowledge only implicitly contained in the knowledge base. One could argue that in the context of evolution of description logic knowledge bases it is natural to have this distinction. Nevertheless there is a huge body of research trying to apply the AGM theory to description logic knowledge bases with many different, partially discordant approaches how to reformulate the AGM postulates, some of them even coming to the conclusion that the AGM theory is not suitable for description logics (Flouris, Plexousakis, and Antoniou, 2005). See (Flouris, Huang, Pan, Plexousakis, and

Wache, 2006), (Flouris, Plexousakis, and Antoniou, 2004), (Qi, Liu, and Bell, 2006), and (Qi and Yang, 2008) for a survey on different revision approaches. In this chapter we neither introduce a new set of postulates nor try to develop revision operators complying an already announced set of postulates. Instead we define three operations for the evolution of the instance level of an ontology, namely insertion of assertions, deletion of assertions and repair of inconsistent. These operations are defined such that it is ensured that the changes applied to the instance level of the knowledge base are minimal. The focus of this chapter on the method how to compute these operations.

In Section 4.1, an approach to compute view deletion in deductive databases is introduced and it is shown how the problem of deleting atoms from a deductive database can be turned into a proof task which can be tackled by a theorem prover. Section 4.2 addresses the problem of evolution of the instance level of knowledge bases given in the description logic $\mathcal{SHI}$. For this, Section 4.2.1 introduces the different operations to revise ABoxes. Section 4.2.2 describes a transformation of the description logic knowledge base, the so-called $\mathcal{K}^*$-transformation which forms the basis for the computation of the evolution of ABoxes presented in Section 4.2.3. The approach presented is implemented and Section 4.2.4 provides experimental results. The chapter closes with a survey of selected related work in Section 4.3 together with an outlook on future work in Section 4.4.

4.1 View Deletion in Deductive Database

Baumgartner, Fröhlich, Furbach, and Nejdl (1997) present an approach for model based diagnosis using theorem proving techniques. Baumgartner et al. (1997) use a similar technique for the computation of view deletion in deductive databases. We will present the basic ideas of this approach in this section. For this, we first present the notion of deductive databases and view deletion. Then, in Section 4.1.1, an algorithm to efficiently compute view deletion in deductive databases is presented.

A *definite deductive database* (*DDB*) consists of two parts: the *intensional database* (*IDB*) and the *extensional database* (*EDB*). The *IDB* is the immutable part of the *DDB* and consists of definite clauses, meaning horn clauses with exactly one positive literal. In contrast to this, the *EDB* constitutes the updatable part and consists of ground facts. The meaning of the *DDB* is given by least herbrand model semantics. According to Lloyd (1993), the least Herbrand model for an *IDB* is the set of all ground atoms which are a logical consequence of the *IDB*. Baumgartner et al. (1997) considers first-order logic without function symbols meaning that the Herbrand Base is finite. In this case the clauses given in the *IDB* can be seen as an abbreviation to the ground clauses obtained by replacing all variables in the clauses by terms of the Herbrand universe in all possible ways. Therefore the *IDB* can be seen as a set of ground clauses which corresponds to a set of propositional logic clauses. This is why in the following, the *IDB* is assumed to be ground and all examples provided in this section contain propositional logic clauses. A *view predicate* is a predicate defined in the *IDB* by occurring in the head of a clause. A *view atom* is an atom with a view predicate. Similar to the notion of view predicates and view atoms, the terms base predicate and base atoms are defined. A *base predicate*

is a predicate defined in the *EDB* and a *base atom* is an atom with a base predicate. Baumgartner et al. (1997) assumes that the *IDB* does not contain any unit clauses and there is no predicate which is both view and atom.

In the literature it is common to write clauses of a *DDB* as $a \leftarrow b_1 \wedge \ldots \wedge b_n$ with the implication arrow pointing to the left. For the sake of standardization of notation throughout the thesis at hand, we deviate from this notation and write the implication arrow pointing to the right.

Example 4.1.1. *Consider the DDB consisting of the following IDB and EDB:*

$$\begin{array}{lll} \textit{IDB:} & a \wedge b \rightarrow c & \textit{EDB:} \quad \rightarrow b \\ & d \rightarrow c & \qquad\quad \rightarrow d \\ & d \rightarrow e & \\ & f \rightarrow a & \end{array}$$

The set of view atoms in DDB is $\{a, e, c\}$ *and the set of base atoms is* $\{b, d\}$.

The *view deletion problem* describes the task to delete an atom which follows from a *DDB* by removing as little facts as possible from the *EDB* such that afterwards the atom is not contained in the deductive closure anymore. A subset of the *EDB* which together with the *IDB* implies an atom b is called *explanation* for b.

A straightforward idea to perform a deletion is to first compute all explanations for the atom supposed to be deleted and then to remove at least one element from each of these explanations. This corresponds to constructing a hitting set of the explanations for the atom which has to be deleted. Intuitively, a hitting set of a set of sets $\{S_1, \ldots, S_n\}$ is a set containing at least one element from each of the sets S_i, $1 \leq i \leq n$.

Definition 4.1.2 (Hitting Set)**.** *Let* $S = \{S_1, \ldots, S_n\}$ *be a set of sets. The set HS is a hitting set of S iff* $HS \subseteq \bigcup_{i=1}^{n} S_i$ *and for every* $i \in \{1, \ldots n\}$ *with* $S_i \neq \emptyset$, $HS \cap S_i \neq \emptyset$.

Aravindan and Dung (1995) performs deletion with an algorithm using the idea of constructing a hitting set as described above. To delete an atom b, the algorithm first computes the set of all subsets which are explanations for b and then constructs a hitting set of these explanations. This hitting set is then deleted from the *EDB* and the algorithm returns the resulting *DDB*. This approach seems to be natural but has the disadvantage that all explanations of b have to be computed and kept in memory. This leads to an exponential space complexity.

In contrast to this the algorithm presented by Baumgartner et al. (1997), which we will present in Section 4.1.1 does not compute all explanations but directly computes a hitting set. The construction of this hitting set is done by the hyper tableau calculus in a goal oriented way.

Before we consider details of this algorithm, it is interesting to point out a similarity to description logic knowledge bases: The division of a *DDB* into an immutable and an updatable part lends itself to description logic knowledge bases as well. In description logic the knowledge base consists of a TBox, an RBox and an ABox. For many applications the TBox and the RBox are constructed once and rarely changed afterwards.

In contrast to that, the ABox, containing assertions about the individuals, represents a certain state of the world and is subject to frequent changes. Because of this similarity of description logic knowledge bases to deductive databases, techniques developed for the view deletion in deductive databases are potentially helpful for the deletion in ABoxes as well. The technique used for the evolution of $\mathcal{SHI}$ ABoxes introduced by Furbach and Schon (2013a) and presented in the next section can be seen as a lifting of the technique introduced by Baumgartner et al. (1997) to description logics. Before this technique is presented in detail, a short overview on the techniques used by Baumgartner et al. (1997) follows.

4.1.1 Using Theorem Proving Techniques for Deletion in Deductive Databases

Aravindan and Baumgartner (2000) presents an approach for view deletion in deductive databases. Compared to the algorithm introduced by Aravindan and Dung (1995) which presented in the previous section, this algorithm has the advantage that it is not necessary to compute all possible explanations for the atom which is to be deleted. The approach is based on the assumption that when deleting an atom from a *DDB* most likely only a very small fraction of the *EDB* has to be deleted whereas the majority of the facts in the *EDB* are preserved. This is why the *EDB* is considered to be close to a model for the *IDB* and only the deviation of the *EDB* is computed. The technique used by Aravindan and Baumgartner (2000) consists of a syntactical transformation of the *DDB* called *renaming*. This technique was first used by Baumgartner et al. (1997) for model based diagnosis. The result of this transformation is combined with information on the atom supposed to be deleted. Then a hyper tableau is constructed and minimal models in this hyper tableau provide minimal deletions.

In the algorithm introduced by Aravindan and Baumgartner (2000), atoms are renamed. For this we introduce a function called Neg.

Definition 4.1.3 (Neg function)**.** *Let p be a predicate symbol and t_i terms, $1 \leq i \leq n$. The* Neg *function maps atoms to renamed atoms:*

$$\mathsf{Neg}(p(t_1, \ldots t_n)) = Negp(t_1, \ldots t_n)$$

We give a slightly modified version of the definition of renaming presented by Aravindan and Baumgartner (2000). Please note that in the following definition a conjunction is interpreted as the set of its conjuncts and a disjunction is interpreted as the set of its disjuncts.

Definition 4.1.4 (Renaming)**.** *Let $\mathbf{B}$ be a conjunction of atoms, $\mathbf{H}$ a disjunction of atoms and $C = \mathbf{B} \rightarrow \mathbf{H}$ a clause with signature Σ, and let $S \subseteq \Sigma$ be a set of ground atoms. Then $R_S(C)$, the* renaming of *C w.r.t. S is*

$$R_S(C) = (\bigwedge_{B \in \mathbf{B} \setminus S} B) \wedge (\bigwedge_{A \in \mathbf{H} \cap S} \mathsf{Neg}(A)) \rightarrow (\bigvee_{A \in \mathbf{H} \setminus S} A) \vee (\bigvee_{B \in \mathbf{B} \cap S} \mathsf{Neg}(B))$$

For a set clauses N, the renaming $R_S(N)$ w.r.t. S is defined as the union of the renaming of all its clauses.

The renaming of a clause w.r.t. a subset of its signature S is the result of first changing the names of all atoms B occurring in S to $NegB$ and then making them switch to the other side of the implication. For a special set S_0 constructed from all ground atoms occurring in the EDB together with the atoms occurring in the head of a clause in the IDB, renaming w.r.t. S_0 is called IDB^* transformation.

Definition 4.1.5 (IDB^* Transformation (Aravindan and Baumgartner, 2000))**.** *Let $DDB = IDB \cup EDB$ be a deductive data base and S_0 a set of ground atoms given as*

$$S_0 = EDB \cup \{A \mid A \text{ occurs in the head of a clause of the } IDB\}.$$

Then IDB^* is the renaming of IDB w.r.t. S_0.

Example 4.1.6. *Consider the DDB given in Example 4.1.1 The set of atoms S_0 from Definition 4.1.5 is $S_0 = \{a, b, c, d, e\}$. Renaming IDB w.r.t. S_0 leads to*

$$\begin{aligned} IDB^*: \quad & Negc \rightarrow Nega \vee Negb \\ & Negc \rightarrow Negd \\ & Nege \rightarrow Negd \\ f \wedge Nega & \rightarrow \end{aligned}$$

Given an interpretation $\mathcal{I}$ for a clause C, it is possible to construct a model $\mathcal{I}^S$ from $\mathcal{I}$ such that $\mathcal{I}^S \models R_S(C)$ iff $\mathcal{I} \models C$. This can be done by setting $\mathcal{I}^S(NegB) = true$ iff $\mathcal{I}(B) = false$ for $B \in S$ and $\mathcal{I}^S(B) = true$ iff $\mathcal{I}(B) = true$ for $B \notin S$.

The IDB^* transformation can be used to efficiently delete atoms from the deductive closure of a DDB. Intuitively, an atom $NegB$ occurring in a model is interpreted as the deletion of atom B. Performing a request to delete atom B can be done by adding $NegB$ to IDB^*. Using a bottom-up reasoning procedure like the hyper tableau calculus after adding the fact $NegB$ leads to a goal directed behavior which makes this calculus a good choice for the construction of models for $IDB^* \cup \{\rightarrow NegB\}$. Models can be read from open and finished branches of a hyper tableau. The intuition is that those Neg-atoms occurring in a model represent a hitting set. Therefore removing those atoms from the EDB puts the deletion into effect. This hitting set is constructed not only in a goal oriented way but further without constructing all possible explanations.

Definition 4.1.7 (Update Tableaux (Aravindan and Baumgartner, 2000))**.** *Let $DDB = IDB \cup EDB$ be a deductive database from which $B \in EDB$ should be deleted. An update tableau T for DDB and the delete request B is a hyper tableau for $IDB^* \cup \{\rightarrow NegB\}$ such that every open branch is finished. The hitting set of an open branch b in T is defined as*

$$HS(b) = \{A \in EDB \mid \mathsf{Neg}(A) \in b\}.$$

Update tableaux can be used for the computation of deletions. Aravindan and Baumgartner (2000) proves completeness of update tableaux. Completeness means that for a $DDB = EDB \cup IDB$ and a delete request B and a minimal set $\alpha \subseteq EDB$ such that $IDB \cup (EDB \setminus \alpha) \not\models B$, there is an open finished branch b in every update tableau for DDB and B such that $HS(b) = \alpha$.

Correctness of update tableaux means that for every open finished branch b in an update tableau, there is a minimal set $\alpha \subseteq EDB$ such that $IDB \cup (EDB \setminus \alpha) \not\models B$ and $HS(b) = \alpha$. To get this property, it is necessary to consider only those branches of the update tableau, representing a model which is minimal w.r.t. the set of *Neg*-atoms produced by renaming.

Definition 4.1.8 (Branch Satisfying Minimality (Aravindan and Baumgartner, 2000))**.** *Let $DDB = IDB \cup EDB$ be a deductive database from which ground atom $B \in EDB$ should be deleted. Let further T be an update tableau for DDB and the delete request B and b an open finished branch in T. Branch b satisfies minimality, iff*

$$\forall s \in HS(b) : (IDB \cup EDB \setminus HS(b) \cup \{s\}) \models B.$$

The following algorithm can be used for view deletion in deductive databases:

Algorithm 4.1.9 (View Deletion Algorithm Based on the Minimality Test (Aravindan and Baumgartner, 2000))**.**

Input: *A database $DDB = IDB \cup EDB$ and a ground atom B to be deleted.*
Output: *A new database $DDB' = IDB \cup EDB'$*

begin

1. *Construct an update tableau T for DDB and delete request B.*
2. *Find a branch b in T satisfying minimality.*
3. *Return $IDB \cup (EDB \setminus HS(b))$.*

end

Proofs of both correctness and completeness of Algorithm 4.1.9 are provided by Aravindan and Baumgartner (2000). The algorithm only considers one branch at a time. And even the minimality test can be done without considering other branches. Due to regularity of the update tableau, the size of a branch is bounded by the size of the signature of DDB. Therefore, the space complexity of this algorithm is polynomial w.r.t. the size of the signature of DDB. The time complexity of the algorithm is Π_2^p, since minimal model computation is known to be in this complexity class (Eiter and Gottlob, 1993). In the second step of the algorithm it is necessary to determine a branch satisfying minimality. The minimality test used for this corresponds to the groundednesstest as introduced by Niemelä (1996b) and Niemelä (1996a). The basic idea of the groundedness test is to check if the elements occurring in the model are implied by the IDB together with the negation of the atoms which are not present in the model. The following definition adapts the groundedness test to the task of view deletion in deductive databases.

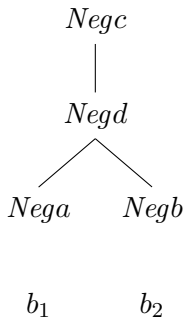


Figure 4.1: Update Tableau for $IDB^* \cup \{\rightarrow Negc\}$ Containing Two Branches b_1 and b_2.

Definition 4.1.10 (Groundedness Test (Aravindan and Baumgartner, 2000)). *Let $DDB = EDB \cup IDB$ be a deductive database from which ground atom B is supposed to be a deleted and T be an update tableau for DDB and B. An open finished branch b satisfies the groundedness test, iff*

$$\forall s \in HS(b) : IDB^* \cup \{NegD \mid D \in (EDB \setminus HS(b))\} \cup \{\rightarrow NegB\} \vdash\!\!\rightarrow \mathsf{Neg}(s)$$

According to Proposition 5.1 by Aravindan and Baumgartner (2000) both the minimality test and the groundedness test can be used to find out if a model is a minimal model: a model is a minimal model iff it satisfies minimality and the groundedness test. Both the minimality test and the groundedness test involve testing if an implication holds for all elements of the computed hitting set. This can be turned into a satisfiability test which can be carried out by any refutational prover. Given a deductive database $DDB = EDB \cup IDB$ from which ground atom B is supposed to be a deleted and T an update tableau for DDB and B with b an open finished branch in T. A branch b satisfies minimality, iff

$$(IDB \cup (EDB \setminus HS(b)) \cup \{B \rightarrow\}) \wedge \bigvee_{s \in HS(b)} s$$

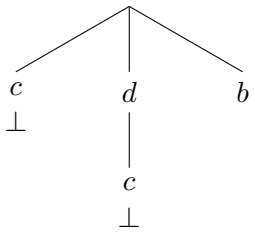
is unsatisfiable, and is grounded iff

$$(IDB^* \cup \{\mathsf{Neg}(D) \mid D \in (EDB \setminus HS(b))\} \cup \{\rightarrow NegB\}) \wedge \bigvee_{s \in HS(b)} \neg\mathsf{Neg}(s)$$

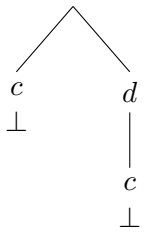
is unsatisfiable.

Example 4.1.11. *Reconsider the DDB presented in Example 4.1.1. Now atom c is supposed to be deleted. For this we add $\{\rightarrow Negc\}$ to IDB^* given in Example 4.1.6.*

The update tableau for $IDB^* \cup \{\rightarrow Negc\}$ is shown in Figure 4.1. The hitting set for the left branch b_1 is $HS(b_1) = \{c, d\}$ and the hitting set for the right branch b_2 is $HS(b_2) = \{c, d, b\}$. Since $HS(b_1) \subset HS(b_2)$, we can conclude that deleting $\{c, d, b\}$ from the EDB is no minimal deletion. However, Algorithm 4.1.9 only considers one branch at a time and therefore does not compare the hitting sets of different branches. In contrast



(a) Hyper Tableau for the Minimality Test for Branch b_2.



(b) Hyper Tableau for the Minimality Test for Branch b_1.

to that, it uses the minimality test to determine if a hitting set is minimal. For branch b_2, the minimality test leads to a satisfiability test of

$$(IDB \cup (EDB \setminus HS(b_2)) \cup \{c \rightarrow\}) \wedge \bigvee_{s \in HS(b_2)} s$$

which corresponds to checking the satisfiability of the following set of clauses

$$\begin{aligned}\{&a \wedge b \rightarrow c,\\ &d \rightarrow c,\\ &d \rightarrow e,\\ &f \rightarrow a,\\ &c \rightarrow,\\ &\rightarrow c \vee d \vee b\}.\end{aligned}$$

The hyper tableau calculus can be used to perform this satisfiability test. The result is shown in Figure 4.2a. Since the tableau is not closed, it can be concluded that branch b_2 does not satisfy the minimality and therefore does not provide a minimal deletion.

In order to test the model provided by branch b_1 for minimality, the satisfiability of the following set of clauses has to be checked

$$\begin{aligned}\{&a \wedge b \rightarrow c,\\ &d \rightarrow c,\\ &d \rightarrow e,\\ &f \rightarrow a,\\ &c \rightarrow,\\ &\rightarrow c \vee d\}.\end{aligned}$$

The hyper tableau for this clause set shown in Figure 4.2b is closed. Therefore, we can conclude that b_1 satisfies minimality and yields to a minimal deletion.

The renaming technique used for view deletion in databases presented in this section was first introduced by Baumgartner et al. (1997) where it was used for diagnosis applications. There a digital circuit is given and the task is to find a minimal set of components whose faultiness explains the circuit's behavior. In this application, it is natural to strive for minimality since it is less likely that a failure is caused by the faultiness of a large number of components. In this setting, the approach was implemented and proved to be very successful.

4.2 Semantically Guided Evolution of $\mathcal{SHI}$ ABoxes

Description logics are broadly used to model knowledge. Since description logics form the basis of some ontology languages such as OIL, DAML+OIL and OWL (Horrocks, Patel-Schneider, and Harmelen, 2003), they play an important role for the Semantic Web as well (Berners-Lee, Hendler, and Lassila, 2001). Furthermore, description logics are used in many applications like bio-informatics and medical terminologies (Stevens, Goble, Horrocks, and Bechhofer, 2002; Rector and Horrocks, 1997). Ontologies used in practice usually are subject to frequent changes. Therefore, algorithms for modeling changes in description logics knowledge bases are of great interest to the Semantic Web community. See (Halaschek-Wiener, Parsia, and Sirin, 2006; Liu, Lutz, Milicic, and Wolter, 2011) for details. On the other hand, changes performed to an ontology can easily introduce inconsistencies, which constitute a problem, since it is possible to deduce everything from an inconsistent ontology. To avoid this undesired behavior, the removal of inconsistencies, often called ontology repair, is a focus of research in the area of description logic (Horridge, Parsia, and Sattler, 2009; Bienvenu and Rosati, 2013a; Rosati, 2011).

This section focuses on the evolution of the instance level, namely the ABox, of a description logic knowledge base. For this, the TBox is considered to be both fixed and consistent and only the ABox is subject to changes. Three different operations on the instance level are considered: *deletion*, *insertion* and *repair*. Instance level deletion addresses the deletion of an ABox assertion from the deductive closure of the knowledge base by removing as few ABox assertions as possible. Instance level insertion means adding an ABox assertion to the knowledge base. In both cases it is important that the resulting ABox is consistent w.r.t. the rest of the knowledge base. For the task of ABox repair, an ABox inconsistent w.r.t. its TBox and RBox is given. The aim of the repair operation is to remove a minimal set of assertions from the ABox such that the resulting ABox together with the TBox and the RBox is consistent.

Exemplarily, we illustrate the task of repairing an ABox which is inconsistent w.r.t. its TBox by an example which is taken from Lenzerini and Savo (2011) and adapted to $\mathcal{SHI}$. Consider the knowledge base $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ with empty RBox. The terminological part

of the knowledge base is given by

$$\begin{aligned}\mathcal{T} = \{&\mathit{Mechanic} \sqsubseteq \mathit{TeamMember},\\ &\mathit{Driver} \sqsubseteq \ \mathit{TeamMember},\\ &\mathit{Driver} \sqsubseteq \neg \mathit{Mechanic},\\ &\exists \mathit{drives}.\top \sqsubseteq \mathit{Driver}\}.\end{aligned}$$

According to this TBox, both drivers and mechanics are team members, but drivers are not mechanics. Furthermore, $\mathcal{T}$ specifies that someone driving something is a driver. The assertional knowledge is given by the following ABox.

$$\begin{aligned}\mathcal{A} = \{&\mathit{Driver}(\mathit{felipe}),\\ &\mathit{Mechanic}(\mathit{felipe}),\\ &\mathit{TeamMember}(\mathit{felipe}),\\ &\mathit{drives}(\mathit{felipe}, \mathit{ferrari})\}.\end{aligned}$$

It is easy to see that $\mathcal{A}$ is inconsistent w.r.t. $\mathcal{T}$, because *felipe* belongs to the concepts *Driver* and *Mechanic* which is forbidden by the third axiom of $\mathcal{T}$. The task is now to minimally repair $\mathcal{A}$ meaning to delete a minimal set of ABox assertions such that the resulting ABox is consistent w.r.t. $\mathcal{T}$. There are two possibilities to minimally repair $\mathcal{A}$. The first one is to remove $\mathit{Mechanic}(\mathit{felipe})$ from the ABox. The resulting ABox is

$$\begin{aligned}\mathcal{A}' = \{&\mathit{Driver}(\mathit{felipe}),\\ &\mathit{TeamMember}(\mathit{felipe}),\\ &\mathit{drives}(\mathit{felipe}, \mathit{ferrari})\}.\end{aligned}$$

For the second possibility both $\mathit{Driver}(\mathit{felipe})$ and $\mathit{drives}(\mathit{felipe}, \mathit{ferrari})$ have to be deleted. The removal of $\mathit{drives}(\mathit{felipe}, \mathit{ferrari})$ is necessary, because the third axiom of the TBox would cause $\mathit{Driver}(\mathit{felipe})$ to be still contained in the deductive closure. In this case, the resulting ABox is

$$\begin{aligned}\mathcal{A}'' = \{&\mathit{Mechanic}(\mathit{felipe}),\\ &\mathit{TeamMember}(\mathit{felipe})\}.\end{aligned}$$

Please note that in both cases only those assertions were removed which necessarily had to be removed in order to resolve the inconsistency.

One possibility to compute minimal repairs is to first determine every minimal subset $\mathcal{A}'$ of the ABox $\mathcal{A}$ such that $\mathcal{A}'$ is inconsistent w.r.t. the TBox and RBox and then remove a minimal set of assertions from $\mathcal{A}$ such that exactly one assertion is removed from each subset $\mathcal{A}'$. This corresponds to computing a minimal hitting set for the set of all minimal subsets $\mathcal{A}'$ of $\mathcal{A}$ with $\mathcal{A}'$ inconsistent w.r.t. the TBox and RBox. However, the drawback of this approach is that it requires all minimal subsets $\mathcal{A}'$ to be constructed.

In contrast to this, we present an approach where the minimal hitting set is computed directly without computing all minimal subsets $\mathcal{A}'$. To accomplish this we suggest a

transformation, called $\mathcal{K}^*$-transformation, on the set of DL-clauses corresponding to the knowledge base under consideration. This transformation adapts the idea of the IDB^* transformation introduced in Definition 4.1.5 in Section 4.1.1 to description logic knowledge bases. In Section 4.1.1, the transformation is guided by a set of ground atoms likely to be a model for the deductive database under consideration. Similar to this, the $\mathcal{K}^*$-transformation we are about to introduce, is guided by the assertions contained in the ABox, assuming that an ABox can be seen as a (partial) model of the TBox and RBox. One can say that the $\mathcal{K}^*$-transformation is *semantically guided* by the ABox. The advantage of assuming the assertions in the ABox to be true is that the computation of instance level evolution thereby only requires to determine the deviation from the ABox. When considering large ABoxes, it is reasonable to assume that deleting an assertion or repairing such an ABox only results in deleting a very small part of the ABox. In cases where this assumption applies, the approach to only compute the deviation from the ABox is beneficial.

Section 4.2.1 introduces the three different operations for ABox evolution, namely deletion, insertion and repair. Section 4.2.2 presents the $\mathcal{K}^*$-transformation together with some properties of the $\mathcal{K}^*$-transformation with regard to models. Section 4.2.3 shows how to use the $\mathcal{K}^*$-transformation to compute the introduced operations for ABox evolution. The approach is implemented and Section 4.2.4 presents experimental results.

Most parts of this chapter have been published in (Furbach and Schon, 2013a) and (Furbach and Schon, 2013b).

4.2.1 ABox Evolution

The three different operations addressed in our approach are deletion, repair and insertion. In all cases, only the instance level of the knowledge base, namely the ABox, is changed and both the TBox and the RBox are assumed to be fixed and consistent. In addition to that, it is assumed that the ABox only consists of assertions of the form $A(a)$ or $R(a, b)$ with A an atomic concept, R a role and a, b individuals. This assumption is not a restriction since it is possible to transform every $\mathcal{SHI}$ ABox into this form by introducing new concepts. For example an assertion $(A \sqcup B)(a)$ can be translated into $D(a)$ together with the TBox axioms $D \sqsubseteq A \sqcup B$ and $A \sqcup B \sqsubseteq D$ for a new concept D.

In the following, DL-clauses will be used to compute the result of these operations. This is why all examples in this section present knowledge bases in form of DL-clauses. Furthermore, by the term *deductive closure* of a knowledge base $\mathcal{K}$ we denote the set of ABox assertions, which can be deduced from $\mathcal{K}$.

4.2.1.1 Deletion

The first instance level operation we address is deletion of an assertion from an ABox. More specifically the task is as follows: Given a knowledge base $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ with consistent $\mathcal{T} \cup \mathcal{R}$ and an ABox assertion D, remove as few assertions as possible such that D is not contained in the deductive closure of the knowledge base anymore. In general it is not sufficient to only remove D from the ABox because D can still be contained in the

deductive closure. Hence the task is to determine a set of ABox assertions, which have to be deleted from the ABox in order to prevent D from being a logical consequence of the knowledge base. Since minimal deletion is desired, the set of ABox assertions to be deleted is supposed to be minimal. These considerations are reflected in the following definition.

Definition 4.2.1. *(Minimal Instance Deletion) Let $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ be a knowledge base with $\mathcal{R} \cup \mathcal{T}$ satisfiable. Let $D \in \mathcal{A}$ be a ground atom called delete request. Then $\mathcal{A}' \subseteq \mathcal{A}$ is called* minimal instance deletion *of D from $\mathcal{A}$ if $\mathcal{R} \cup \mathcal{T} \cup \mathcal{A}' \not\models D$ and there is no $\mathcal{A}''$ with $\mathcal{A}' \subset \mathcal{A}'' \subseteq \mathcal{A}$ and $\mathcal{R} \cup \mathcal{T} \cup \mathcal{A}'' \not\models D$.*

Note that, due to our assumptions on the structure of the ABox, each delete request D is of the form $A(a)$ or $R(a, b)$. If it is clear from the context that we are talking about instance level deletion, we omit instance level and talk about deletion instead.

Example 4.2.2. *We consider the knowledge base $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ with an empty RBox and with the TBox and the corresponding set of DL-clauses $\Xi(\mathcal{T})$.*

$$\begin{aligned} \mathcal{T} = \{&\exists R.C \sqsubseteq D, \\ &B \sqsubseteq \exists R.C, \\ &D \sqsubseteq C\} \end{aligned} \qquad \begin{aligned} \Xi(\mathcal{T}) = \{&R(x,y) \wedge C(y) \rightarrow D(x), \\ &B(x) \rightarrow \exists R.C(x), \\ &D(x) \rightarrow C(x)\} \end{aligned}$$

The ABox is given as

$$\mathcal{A} = \{B(a),\ D(a),\ R(a,a),\ C(b),\ R(b,b)\}.$$

The delete request $D(a)$ has a minimal instance deletion

$$\mathcal{A}' = \{R(a,a),\ C(b),\ R(b,b)\}$$

4.2.1.2 Repair

There are many possibilities how an ABox can become inconsistent w.r.t. its TBox and RBox. One example is the insertion of new assertions into an ABox. In all cases the goal is to minimally repair the ABox by deleting assertions from the ABox such that the resulting ABox is consistent w.r.t. the TBox and RBox. The term minimally means that only those assertions are removed from the ABox which necessarily have to be removed in order to resolve the inconsistency.

Definition 4.2.3. *(Minimal ABox Repair) Let $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ be a knowledge base where $\mathcal{T} \cup \mathcal{R}$ is satisfiable. $\mathcal{A}' \subseteq \mathcal{A}$ is called* minimal ABox repair *of $\mathcal{A}$ if $\mathcal{R} \cup \mathcal{T} \cup \mathcal{A}'$ is consistent and there is no $\mathcal{A}''$ with $\mathcal{A}' \subset \mathcal{A}'' \subseteq \mathcal{A}$ and $\mathcal{R} \cup \mathcal{T} \cup \mathcal{A}''$ consistent. Further $Rep(\mathcal{K})$ denotes the set of all possible repairs of $\mathcal{K}$.*

Note that we define the notion of a minimal ABox repair in a way, that it is also applicable to an ABox which is consistent to its TBox and its RBox. In this case, the minimal ABox repair corresponds to the original ABox.

Example 4.2.4. *Consider the knowledge base* $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ *with an empty RBox and the TBox*

$$\mathcal{T} = \{C \sqsubseteq \neg D\}$$

which corresponds to the following set of DL-clauses

$$\Xi(\mathcal{T}) = \{\bot \leftarrow C(x) \wedge D(x)\}$$

together with the ABox

$$\mathcal{A} = \{C(a),\, D(a)\}.$$

Clearly $\mathcal{A}$ *is inconsistent w.r.t* $\mathcal{T}$*. There are two different minimal ABox repairs:*

$$\mathcal{A}' = \{C(a)\}$$
$$\mathcal{A}'' = \{D(a)\}.$$

4.2.1.3 Insertion

The third instance level operation we address is insertion of an assertion into an existing ABox. The problem that arises when considering insertion is, that the resulting ABox might be inconsistent w.r.t. the TBox and the RBox.

Example 4.2.5. *Given the following knowledge base* $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ *with an empty RBox and the TBox*

$$\mathcal{T} = \{C \sqsubseteq \neg D\}$$

which corresponds to the following set of DL-clauses

$$\Xi(\mathcal{T}) = \{\bot \leftarrow C(x) \wedge D(x)\}$$

together with the ABox

$$\mathcal{A} = \{C(a)\}.$$

Adding the assertion $D(a)$ *into* $\mathcal{A}$ *leads to*

$$\mathcal{A}' = \{C(a), D(a)\}$$

which is inconsistent w.r.t. $\mathcal{T}$*.*

The following definition of minimal instance insertion is chosen such that inconsistent results are avoided.

Definition 4.2.6. *(Minimal Instance Insertion) Let* $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ *be a knowledge base with* $\mathcal{R} \cup \mathcal{T}$ *consistent and* D *a ground atom of the form* $A(a)$ *or* $R(a, b)$*. An ABox* $\mathcal{A}'$ *is called* minimal instance insertion *of* D *into* $\mathcal{A}$ *iff*

- $D \in \mathcal{A}'$,
- $(\mathcal{A}' \setminus D) \subseteq \mathcal{A}$,

- *$\mathcal{R} \cup \mathcal{T} \cup \mathcal{A}'$ is consistent and there is no $\mathcal{A}''$ with $D \in \mathcal{A}''$ and $(\mathcal{A}' \setminus D) \subset (\mathcal{A}'' \setminus D) \subseteq \mathcal{A}$ and $\mathcal{R} \cup \mathcal{T} \cup \mathcal{A}''$ is consistent.*

If it is clear from the context that we are talking about instance level insertion, we omit instance level and talk about minimal insertion instead.

Example 4.2.7. *A minimal instance insertion of $D(a)$ into the DL-clauses given in Example 4.2.5 is $\mathcal{A}'' = \{D(a)\}$.*

In the following section, a transformation called $\mathcal{K}^*$-transformation is introduced which turns out to be very helpful in order to compute minimal deletion, insertion and repair.

4.2.2 $\mathcal{K}^*$-Transformation

The tasks introduced in Section 4.2.1 can be solved with the help of the so-called $\mathcal{K}^*$-*transformation* which will be introduced in this section. As discussed in the introduction, the transformation will be guided by the ABox of the knowledge base which is regarded as a partial model.

First, minimal instance deletion will be addressed. When deleting a given instance, the task is to determine a minimal set of ABox assertions which have to be removed in order to prevent the instance from being contained in the deductive closure of the knowledge base. The idea of the $\mathcal{K}^*$-transformation is similar to the techniques summarized in Section 4.1. Each occurrence of an atom $A(a)$ in a clause is replaced by $\neg NegA(a)$. This transformation can be seen as atoms switching the sides in the clause representation of DL-clauses. For example in the clause

$$\{\rightarrow A(a)\}$$

atom $A(a)$ is replaced by $\neg NegA(a)$ leading to the clause

$$\{\rightarrow \neg NegA(a)\}$$

corresponding to the clause

$$\{NegA(a) \rightarrow\}.$$

This makes sense when a bottom-up proof procedure like the hyper tableau calculus is used: a fact $\rightarrow A(a)$ changes the side and the clause becomes $NegA(a) \rightarrow$. As a consequence $A(a)$ is not derived explicitly. It is assumed to be in the model unless the opposite, namely $\rightarrow NegA(a)$, is derived.

Deducing an atom $NegA(a)$ means that the ABox has to be revised such that atom $A(a)$ is removed from the ABox. The big advantage of this transformation is that only those atoms which have to be deleted are computed. All remaining atoms will be kept in the ABox. Since it is reasonable to expect the ABox to be very large, it is advantageous to compute only the deviation from the original ABox.

To rename atoms, we extend the Neg function introduced in Definition 4.1.3.

Definition 4.2.8 (Neg function)**.** *The* Neg *function maps atoms to renamed atoms:*

- *For atomic concepts A and an individual or variable a:*

$$\mathsf{Neg}(A(a)) = NegA(a).$$

- *For atomic roles R and individuals or variables a, b:*

$$\mathsf{Neg}(R(a,b)) = NegR(a,b).$$

Further, for a set of atoms P, $\mathsf{Neg}(P)$ *is defined as:* $\mathsf{Neg}(P) = \{\mathsf{Neg}(A) \mid A \in P\}$.

We slightly abuse notation by using the Neg function to rename atomic concepts and atomic roles: for B an atomic concept or an atomic role: $\mathsf{Neg}(B) = NegB$. So we can use the Neg function to rename atoms, sets of atoms and atomic concepts and roles.

Note that $\mathsf{Neg}(C(a)) = NegC(a)$ and we will use both interchangeably.

In the following it is convenient to have a function extracting the atomic concept / atomic role from an atom:

Definition 4.2.9. *(Symbol Extraction Function) Let A be an atom. Then $\sigma(A)$ is defined as follows:*

$$\sigma(A) = \begin{cases} B & \text{if } A = B(s) \text{ for some atomic concept } B, \\ R & \text{if } A = R(s,t) \text{ for some atomic role } R, \\ \exists R.B & \text{if } A = \exists R.B(s) \text{ for some role name } R, \\ & \text{and } B = E \text{ or } \neg E \text{ for some atomic concept } E. \end{cases}$$

By $\sigma(N)$ for a set of atoms N we denote the union of $\sigma(A)$ for all atoms $A \in N$.

As mentioned before, we want to use the idea of the approach for view deletion in deductive databases introduced in Section 4.1.1 for the computation of ABox evolution in the description logic $\mathcal{SHI}$. The next definition extends the idea of renaming introduced in Definition 4.1.4 to DL-clauses. Note that in the following definition, the body **B** of a DL-clause is treated as the set of its conjuncts and the head **H** is treated as the set of its disjuncts.

Definition 4.2.10 (Renaming)**.** *Let DL be a set of DL-clauses and S a set of atomic concepts and atomic roles. Let $C \in DL$ be $C = \mathbf{B} \rightarrow \mathbf{H}$. Then $R_S(C)$, the* renaming of *C w.r.t. S is*

$$
\begin{aligned}
&\mathsf{R}_S(C) = \\
&\{C\} && (4.1)\\
&\cup \\
&\{(\bigwedge_{\substack{B\in\mathbf{B},\\ \sigma(B)\notin S}} B) \wedge (\bigwedge_{\substack{A\in\mathbf{H},\\ \sigma(A)\in S}} \mathsf{Neg}(A)) \rightarrow (\bigvee_{\substack{A\in\mathbf{H},\\ \sigma(A)\notin S}} A) \vee (\bigvee_{\substack{B\in\mathbf{B},\\ \sigma(B)\in S}} \mathsf{Neg}(B))\} && (4.2)\\
&\cup \\
&\{R(x,y) \wedge NegR(x,y) \rightarrow \bot \mid \exists A \in (\mathbf{H}\cup\mathbf{B}) \textit{ with } \sigma(A) = R \textit{ and } R \in S \textit{ or} \\
&\qquad\qquad \exists A \in \mathbf{H} \textit{ of the form } A = \exists R.C(z) \textit{ and } R \in S\} && (4.3)\\
&\cup \\
&\{D(x) \wedge NegD(x) \rightarrow \bot \mid \exists A \in (\mathbf{H}\cup\mathbf{B}) \textit{ with } \sigma(A) = D \textit{ and } D \in S \textit{ or} \\
&\qquad\qquad \exists A \in \mathbf{H} \textit{ of the form } A = \exists R.D(z) \textit{ and } R \in S\} && (4.4)
\end{aligned}
$$

For a set of DL-clauses DL, the renaming $\mathsf{R}_S(DL)$ w.r.t. S is defined as the union of the renaming of all its clauses.

Note that renaming is a bijective function on a set of DL-clauses. Further, renaming can be performed in time linear to the size of the set of DL-clauses times the size of S. Comparing Definition 4.2.10 with the definition of renaming as introduced by Baumgartner et al. (1997) and given in Definition 4.1.4 shows that the basic idea is very similar. In Definition 4.1.4 all atoms occurring in the set S are affected by renaming meaning that they are renamed and change to the other side of the implication sign. Whereas in Definition 4.2.10 all atoms containing a concept or role name occurring in S except existential role restrictions are affected by renaming. Since the construction of DL-clauses is more complicated then the clauses considered in Definition 4.1.4, the extension of renaming to the description logic $\mathcal{SHI}$ produces some more clauses: firstly, the original DL-clause is kept. Secondly for all concept and role symbols affected by renaming additional clauses are added, stating that nothing can belong to a concept C and its renamed version $NegC$. Please note that existential role restrictions are not directly affected by the renaming process. They remain unchanged in the head of a clause even if the respective role restriction or concept is contained in S.

The next proposition states the fact that renaming preserves satisfiability. Furthermore given a model for a set of DL-clauses DL, it is possible to compute a model for the renamed set of DL-clauses $\mathsf{R}_S(DL)$ and vice versa.

Proposition 4.2.11. *(Renaming Models) Let DL be a set of DL-clauses, S a set of atomic concepts and atomic roles and $\mathcal{I}$ an interpretation. Let further $\mathcal{I}^S$ be an interpretation such that $\mathcal{I}^S$ and $\mathcal{I}$ have the same domain and the same interpretation of individuals. In addition to that the interpretation of all roles and concepts occurring in DL coincide. Further $(\mathsf{Neg}(B))^{\mathcal{I}^S} = \overline{B^{\mathcal{I}}}$ for all concepts names $B \in S$ and $(\mathsf{Neg}(R))^{\mathcal{I}^S} = \overline{R^{\mathcal{I}}}$ for all atomic roles $R \in S$. Then $\mathcal{I} \models DL$ iff $\mathcal{I}^S \models \mathsf{R}_S(DL)$.*

Proof. The assumptions of the propositions hold.

1. We first show the direction that $\mathcal{I} \models DL$ implies $\mathcal{I}^S \models \mathsf{R}_S(DL)$. Let $\mathcal{I} \models DL$. Then $\mathcal{I} \models D$ for all clauses D in DL. Let us now assume that $\mathcal{I}^S \not\models \mathsf{R}_S(DL)$. Note that for concepts and roles in $B \in S$, $B^{\mathcal{I}^S} = B^{\mathcal{I}}$ and $(\mathsf{Neg}(B))^{\mathcal{I}^S}$ are complementary. Hence it is obvious that $\mathcal{I} \models DL$ implies $\mathcal{I}^S \models C^S$ for clauses $C^S \in \mathsf{R}_S(DL)$ created by (4.3) and (4.4) of Definition 4.2.10. Further clause C added to $\mathsf{R}_S(DL)$ by (4.1) of Definition 4.2.10 is in DL and therefore $\mathcal{I} \models C$. Since the interpretation of all concept and role symbols in DL coincide for $\mathcal{I}$ and $\mathcal{I}^S$, it follows that $\mathcal{I}^S \models C$. Therefore we can restrict ourselves to clauses C^S in $\mathsf{R}_S(DL)$ created by (4.2) of Definition 4.2.10. Since per assumption $\mathcal{I}^S \not\models \mathsf{R}_S(DL)$, there has to be such a clause C^S with $\mathcal{I}^S \not\models C^S$. This clause C^S is the renaming of some clause $C \in DL$ w.r.t. S. This clause C has the form

$$C = H_1 \vee \ldots \vee H_n \vee A_{h_1} \vee \ldots \vee A_{h_k} \leftarrow B_1 \wedge \ldots \wedge B_m \wedge A_{b_1} \wedge \ldots \wedge A_{b_l}.$$

such that

$$\{\sigma(H_1), \ldots, \sigma(H_n), \sigma(B_1), \ldots, \sigma(B_m)\} \cap S = \emptyset \tag{4.5}$$
$$\{\sigma(A_{h_1}), \ldots, \sigma(A_{h_k}), \sigma(A_{b_1}), \ldots, \sigma(A_{b_l})\} \subseteq S \tag{4.6}$$

Renaming clause C leads to

$$C^S = B_1 \wedge \ldots \wedge B_m \wedge \mathsf{Neg}(A_{h_1}) \wedge \ldots \wedge \mathsf{Neg}(A_{h_k}) \rightarrow H_1 \vee \ldots \vee H_n \vee \mathsf{Neg}(A_{b_1}) \vee \ldots \vee \mathsf{Neg}(A_{b_l}).$$

From $I^S \not\models C^S$ and the definition of the semantics of DL-clauses follows, that there is a variable mapping μ with

$$\mathcal{I}^S, \mu \models B_1 \wedge \ldots \wedge B_m \wedge \mathsf{Neg}(A_{h_1}) \wedge \ldots \wedge \mathsf{Neg}(A_{h_k})$$

and

$$\mathcal{I}^S, \mu \not\models H_1 \vee \ldots \vee H_n \vee \mathsf{Neg}(A_{b_1}) \vee \ldots \vee \mathsf{Neg}(A_{b_l}).$$

It follows that

$$\mathcal{I}^S, \mu \models B_i \text{ for } i \in \{1, \ldots, m\},$$
$$\mathcal{I}^S, \mu \models \mathsf{Neg}(A_{h_i}) \text{ for } i \in \{1, \ldots, k\},$$
$$\mathcal{I}^S, \mu \not\models H_i \text{ for } i \in \{1, \ldots, n\},$$
$$\mathcal{I}^S, \mu \not\models \mathsf{Neg}(A_{b_i}) \text{ for } i \in \{1, \ldots, l\}.$$

From (4.5), (4.6) and the way I^S is constructed, it follows that

$$\mathcal{I}, \mu \models B_i \text{ for } i \in \{1, \ldots, m\},$$
$$\mathcal{I}, \mu \not\models A_{h_i} \text{ for } i \in \{1, \ldots, k\},$$
$$\mathcal{I}, \mu \not\models H_i \text{ for } i \in \{1, \ldots, n\},$$
$$\mathcal{I}, \mu \models A_{b_i} \text{ for } i \in \{1, \ldots, l\}.$$

Therefore

$$\mathcal{I}, \mu \models B_1 \wedge \ldots \wedge B_m \wedge A_{b_1} \wedge \ldots \wedge A_{b_l}$$

and

$$\mathcal{I}, \mu \not\models H_1 \vee \ldots \vee H_n \vee A_{h_1} \vee \ldots \vee A_{h_k}.$$

This implies $\mathcal{I}, \mu \not\models C$ and therefore $\mathcal{I} \not\models C$ which is a contradiction to the assumption that $\mathcal{I} \models DL$ and therefore $\mathcal{I}^S \models C^S$ follows. Since clause C was arbitrarily chosen, this implies $\mathcal{I}^S \models \mathsf{R}_S(DL)$.

2. For the other direction we have to show that $\mathcal{I}^S \models \mathsf{R}_S(DL)$ implies $\mathcal{I} \models DL$. Let us now assume that $\mathcal{I}^S \models \mathsf{R}_S(DL)$ and $\mathcal{I} \not\models DL$. This means that there is a clause $C \in DL$ with $\mathcal{I} \not\models C$. So there is a variable mapping μ with $\mathcal{I}, \mu \not\models C$. However since $DL \subseteq \mathsf{R}_S(DL)$ it follows that $C \in \mathsf{R}_S(DL)$. Since the interpretation of all concept and role symbols in $\mathcal{I}$ and $\mathcal{I}^S$ coincide, this implies $\mathcal{I}^S, \mu \not\models C$ which is a contradiction to $\mathcal{I}^S \models \mathsf{R}_S(DL)$.

□

In the following, it is convenient to have a function extracting the set of individuals from a set of ground atoms.

Definition 4.2.12 (Ind Function)**.** *Let $A(a)$ and $R(a, b)$ be atoms and A, B be sets of ground atoms. The* Ind *function maps sets of ground atoms to sets of individuals and is defined as follows*

- $\mathsf{Ind}(A(a)) = \{a\}$,
- $\mathsf{Ind}(R(a, b)) = \{a, b\}$ *and*
- $\mathsf{Ind}(A \cup B) = \mathsf{Ind}(A) \cup \mathsf{Ind}(B)$.

Definition 4.2.13 ($\mathcal{K}^*$-Transformation)**.** *Let $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ be a knowledge base with satisfiable $\mathcal{R} \cup \mathcal{T}$. Let S be the set of atomic concepts and atomic roles occurring in $\mathcal{A}$ and $\mathcal{T}$. Then $\mathcal{K}^*$ is the clause set obtained by renaming $\Xi(\mathcal{R} \cup \mathcal{T})$ w.r.t. S and adding the DL-clause $\top \to dom(a_1, \ldots, a_n)$ with $\{a_1, \ldots, a_n\} = \mathsf{Ind}(\mathcal{A})$.*

Strictly speaking, $\top \to dom(a_1, \ldots, a_n)$ is not a DL-clause because atoms in DL-clauses are only constructed from unary or binary predicates. We slightly abuse notation here and regard $\top \to dom(a_1, \ldots, a_n)$ as a DL-clause. We could easily avoid this issue by introducing n DL-clauses $\top \to dom(a_i)$, $1 \leq i \leq n$ since the main reason for adding $\top \to dom(a_1, \ldots, a_n)$ with $\{a_1, \ldots, a_n\} = \mathsf{Ind}(\mathcal{A})$ to the result of renaming is to introduce the individuals occurring in the ABox to the result of the $\mathcal{K}^*$-transformation.

Example 4.2.14. *Reconsider the following knowledge base $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ with an empty RBox from Example 4.2.2. The TBox and the corresponding set of DL-clauses $\Xi(\mathcal{T})$ was given as:*

$$\begin{aligned}\mathcal{T} = \{&\exists R.C \sqsubseteq D,\\ &B \sqsubseteq \exists R.C,\\ &D \sqsubseteq C\}\end{aligned} \qquad \begin{aligned}\Xi(\mathcal{T}) = \{&R(x,y) \wedge C(y) \rightarrow D(x),\\ &B(x) \rightarrow \exists R.C(x),\\ &D(x) \rightarrow C(x)\}\end{aligned}$$

Further the following ABox is given:

$$\mathcal{A} = \{B(a),\ D(a),\ C(b),\ R(b,b),\ R(a,a)\}.$$

According to Definition 4.2.13, $S = \{B, D, C, R\}$*. Renaming the DL-clauses* $\Xi(\mathcal{T})$ *w.r.t.* S *leads to the following set of DL-clauses* $\mathcal{K}^*$

$$\begin{aligned} R(x,y) \wedge C(y) &\rightarrow \{D(x),\\ NegD(x) &\rightarrow NegR(x,y) \vee NegC(y),\\ B(x) &\rightarrow \exists R.C(x),\\ \top &\rightarrow \exists R.C(x) \vee NegB(x),\\ D(x) &\rightarrow C(x),\\ NegC(x) &\rightarrow NegD(x),\\ R(x,y) \wedge NegR(x,y) &\rightarrow \bot,\\ C(x) \wedge NegC(x) &\rightarrow \bot,\\ B(x) \wedge NegB(x) &\rightarrow \bot,\\ D(x) \wedge NegD(x) &\rightarrow \bot,\\ \top &\rightarrow dom(a,b)\}. \end{aligned}$$

Proposition 4.2.15. *Let* $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ *be a knowledge base with satisfiable* $\mathcal{R} \cup \mathcal{T}$ *and* S *be the set of atomic concepts and atomic roles occurring in* $\mathcal{A}$ *and* $\mathcal{T}$*. Then* $\Xi(\mathcal{R} \cup \mathcal{T})$*,* $R_S(\Xi(\mathcal{R} \cup \mathcal{T}))$ *and* $\mathcal{K}^*$ *are equisatisfiable.*

Proof. Since $\Xi(\mathcal{R} \cup \mathcal{T}) \subseteq \mathsf{R}_S(\Xi(\mathcal{R} \cup \mathcal{T})) \subseteq \mathcal{K}^*$ holds, unsatisfiability of $\Xi(\mathcal{R} \cup \mathcal{T})$ implies unsatisfiability of both $\mathsf{R}_S(\Xi(\mathcal{R} \cup \mathcal{T}))$ and $\mathcal{K}^*$.

Let M be a model for $\Xi(\mathcal{T} \cup \mathcal{R})$ and $Ind(\mathcal{A}) = \{a_1, \ldots, a_n\}$. According to Proposition 4.2.11 it is possible to construct a model M^S from M with $M^S \models \mathsf{R}_S(\Xi(\mathcal{T} \cup \mathcal{R}))$. We construct a model $M_{\mathcal{K}^*}$ for $\mathcal{K}^*$ by enhancing M^S such that $\top \rightarrow dom(a_1, \ldots, a_n)$ is true. Since the *dom* predicate does not occur in any other DL-clause, it is obvious that $M_{\mathcal{K}^*}$ is a model for $\mathcal{K}^*$. □

Definition 4.2.16 (ground Function)**.** *Let* N_C *be a set of atomic concepts,* N_R *a set of atomic roles and* N_I *be a set of individuals. The* ground *function mapping a set of atomic concepts and atomic roles together with a set of individuals to a set of ground atoms is defined as*

$$\begin{aligned}\mathsf{ground}(N_C \cup N_R, N_I) = &\{C(a) \mid C \in N_C \text{ and } a \in N_I\} \cup\\ &\{R(a,b) \mid R \in N_R \text{ and } a, b \in N_I\}\end{aligned}$$

Lemma 4.2.17 (Model Construction)**.** *Let $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ be a knowledge base with consistent $\mathcal{R} \cup \mathcal{T}$ and S be the set of all concepts and roles occurring in $\mathcal{A}$. Furthermore, let $\mathcal{I}$ be an interpretation. If $\mathcal{I} \models \mathsf{R}_S(\Xi(\mathcal{T} \cup \mathcal{R}))$, then*
$\mathcal{I}' \models \mathsf{R}_S(\Xi(\mathcal{T} \cup \mathcal{R}))$ with $\mathcal{I}' = \mathcal{I} \cup \{A \in \mathsf{ground}(S, \mathsf{Ind}(\mathcal{A})) \mid \mathsf{Neg}(A) \notin \mathcal{I}\}$.

Proof. Let $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ be a knowledge base with consistent $\mathcal{R} \cup \mathcal{T}$, S be the set of all concepts and roles occurring in $\mathcal{A}$ and $\mathcal{I}$ be an interpretation. Furthermore, let DL denote the set of DL-clauses for $\mathcal{T} \cup \mathcal{R}$ meaning that $DL = \Xi(\mathcal{T} \cup \mathcal{R})$.

We proof the lemma by showing that $\mathcal{I} \models \mathsf{R}_S(DL)$ together with $\mathcal{I}' \not\models \mathsf{R}_S(DL)$ leads to a contradiction. Therefore we assume $\mathcal{I} \models \mathsf{R}_S(DL)$ and $\mathcal{I}' \not\models \mathsf{R}_S(DL)$. Then there is a clause $C = (\mathbf{B} \rightarrow \mathbf{H}) \in \mathsf{R}_S(DL)$ with $\mathcal{I} \models C$ but $I' \not\models C$.

This means, that there is a variable mapping μ with $\mathcal{I}, \mu \models C$ and $\mathcal{I}', \mu \not\models C$. For this variable mapping, one of the following two cases has to apply:

1. $\mathcal{I}, \mu \models \mathbf{B}$ and $\mathcal{I}, \mu \models \mathbf{H}$:
   From $\mathcal{I} \subseteq \mathcal{I}'$ follows, that $\mathcal{I}', \mu \models \mathbf{B}$ and $\mathcal{I}', \mu \models \mathbf{H}$, meaning that $\mathcal{I}', \mu \models C$. This is a contradiction to the assumption, that $\mathcal{I}', \mu \not\models C$.

2. $\mathcal{I}, \mu \not\models \mathbf{B}$:
   Clause C has one of these forms:

   (i) $C \in DL$, (4.1) of Definition 4.2.10.

   (ii) C was created by (4.2) of Definition 4.2.10.

   (iii) C was created by (4.3) or (4.4) of Definition 4.2.10.

   We will show that $\mathcal{I}', \mu \models C$ for each of these three subcases, leading to a contradiction to the assumption of $\mathcal{I}', \mu \not\models C$.

   (i) C is an original clause ($C \in DL$, (4.1) of Definition 4.2.10):

   w.l.o.g. C has the form

   $$C = B_1 \wedge \ldots \wedge B_l \wedge B_{l+1} \wedge \ldots \wedge B_n \rightarrow H_1 \vee \ldots \vee H_k \vee H_{k+1} \vee \ldots \vee H_m$$

   with

   $$\{\sigma(H_1), \ldots, \sigma(H_k), \sigma(B_1), \ldots, \sigma(B_l)\} \subseteq S \text{ and}$$
   $$\{\sigma(H_{k+1}), \ldots, \sigma(H_m), \sigma(B_{l+1}), \ldots, \sigma(B_n)\} \cap S = \emptyset.$$

   According to (4.2) of Definition 4.2.10, the following clause C' is added by renaming C w.r.t. S:

   $$C' = \mathsf{Neg}(H_1) \wedge \ldots \wedge \mathsf{Neg}(H_k) \wedge B_{l+1} \wedge \ldots \wedge B_n \rightarrow \mathsf{Neg}(B_1) \vee \ldots \vee \mathsf{Neg}(B_l) \vee H_{k+1} \vee \ldots \vee H_m \quad (4.7)$$

   Since $\{\sigma(B_{l+1}), \ldots, \sigma(B_n)\} \cap S = \emptyset$, it follows that applying μ to the atoms in $\{B_{l+1}, \ldots, B_n\}$ does not lead to ground atoms in $\mathsf{ground}(S, \mathsf{Ind}(\mathcal{A}))$. However the only difference between $\mathcal{I}'$ and $\mathcal{I}$ is in the interpretation of some ground

atoms occurring in $\mathsf{ground}(S, \mathsf{Ind}(\mathcal{A}))$. Therefore $\mathcal{I}$ and $\mathcal{I}'$ interpret the result of applying μ to the atoms in $\{B_{l+1}, \ldots, B_n\}$ in the same way.

We assume $\mathcal{I}', \mu \not\models C$. It follows, that $\mathcal{I}', \mu \models \mathbf{B}$ and $\mathcal{I}', \mu \not\models \mathbf{H}$. This means

$$\mathcal{I}', \mu \models B_1, \ldots, \mathcal{I}', \mu \models B_n. \tag{4.8}$$

As mentioned above, $\mathcal{I}$ and $\mathcal{I}'$ interpret the result of applying μ to the atoms in $\{B_{l+1}, \ldots, B_n\}$ in the same way. This leads to

$$\mathcal{I}, \mu \models B_{l+1}, \ldots, \mathcal{I}, \mu \models B_n. \tag{4.9}$$

According to (4.8), $\mathcal{I}', \mu \models B_j$ for $j \in \{1, \ldots, l\}$. From the way $\mathcal{I}'$ is constructed from $\mathcal{I}$ follows that for each j there are two possible cases:

1. $\mathcal{I}, \mu \models B_j$
2. $\mathcal{I}, \mu \not\models B_j$ and $\mathcal{I}, \mu \not\models \mathsf{Neg}(B_j)$.

In both cases we have $\mathcal{I}, \mu \not\models \mathsf{Neg}(B_j)$, meaning that

$$\mathcal{I}, \mu \not\models \mathsf{Neg}(B_j) \text{ for each } j \in \{1, \ldots, l\}. \tag{4.10}$$

As mentioned above, from $\mathcal{I}', \mu \not\models C$ follows, both $\mathcal{I}', \mu \models \mathbf{B}$ and $\mathcal{I}', \mu \not\models \mathbf{H}$. This means

$$\mathcal{I}', \mu \not\models H_1, \ldots, \mathcal{I}', \mu \not\models H_m. \tag{4.11}$$

$\{\sigma(H_{k+1}), \ldots, \sigma(H_m)\} \cap S = \emptyset$ implies, that appying μ to the atoms in $\{\sigma(H_{k+1}), \ldots, \sigma(H_m)\}$ does not lead to ground atoms which are contained in $\mathsf{ground}(S, \mathsf{Ind}(\mathcal{A}))$. However the only difference between $\mathcal{I}'$ and $\mathcal{I}$ is in the interpretation of ground atoms occurring in $\mathsf{ground}(S, \mathsf{Ind}(\mathcal{A}))$. Therefore the interpretation of the ground atoms resulting from the application of μ to the atoms in $\{\sigma(H_{k+1}), \ldots, \sigma(H_m)\}$ coincides for $\mathcal{I}$ and $\mathcal{I}'$. Therefore

$$\mathcal{I}, \mu \not\models H_{k+1}, \ldots, \mathcal{I}, \mu \not\models H_m.$$

(4.10) together with (4.11) leads to

$$\mathcal{I}, \mu \not\models \mathsf{Neg}(B_1) \vee \ldots \vee \mathsf{Neg}(B_l) \vee H_{k+1} \vee \ldots \vee H_m. \tag{4.12}$$

Meaning, that $\mathcal{I}, \mu$ is not a model for the head of clause C' from (4.7).

According to (4.9), $\mathcal{I}, \mu \models B_{l+1}, \ldots, \mathcal{I}, \mu \models B_n$.

From the way $\mathcal{I}'$ is constructed and the fact that $\{\sigma(H_1), \ldots, \sigma(H_k)\} \subseteq S$ together with (4.11) follows, that

$$\mathcal{I}, \mu \models \mathsf{Neg}(H_1), \ldots, \mathcal{I}, \mu \models \mathsf{Neg}(H_k) \tag{4.13}$$

(4.9) together with (4.13) leads to

$$\mathcal{I}, \mu \models \mathsf{Neg}(H_1) \wedge \ldots \wedge \mathsf{Neg}(H_k) \wedge B_{l+1} \wedge \ldots \wedge B_n. \tag{4.14}$$

Meaning, that $\mathcal{I}, \mu$ is a model for the body of clause C' from (4.7).

Combining (4.12) and (4.14) implies:

$$\mathcal{I}, \mu \not\models C'.$$

Since $C' \in \mathsf{R}_S(DL)$, this is a contradiction to the model property of $\mathcal{I}$.

(ii) C was created by (4.2) of Definition 4.2.10.

$$C = \mathsf{Neg}(H_1) \wedge \ldots \wedge \mathsf{Neg}(H_k) \wedge B_{l+1} \wedge \ldots \wedge B_n \rightarrow \\ \mathsf{Neg}(B_1) \vee \ldots \vee \mathsf{Neg}(B_l) \vee H_{k+1} \vee \ldots \vee H_m$$

with

$$\{\sigma(H_1), \ldots, \sigma(H_k), \sigma(B_1), \ldots, \sigma(B_l)\} \subseteq S \text{ and} \\ \{\sigma(H_{k+1}), \ldots, \sigma(H_m), \sigma(B_{l+1}), \ldots, \sigma(B_n)\} \cap S = \emptyset.$$

Furthermore, $\mathcal{I}, \mu \models C$ and according to assumption (b)

$$\mathcal{I}, \mu \not\models \mathsf{Neg}(H_1) \wedge \ldots \wedge \mathsf{Neg}(H_k) \wedge B_{l+1} \wedge \ldots \wedge B_n.$$

The interpretation of all Neg-atoms coincides for $\mathcal{I}$ and $\mathcal{I}'$. Furthermore $\{\sigma(B_{l+1}), \ldots, \sigma(B_n)\} \cap S = \emptyset$ implies, that $\mathcal{I}$ and $\mathcal{I}'$ interpret $B_{l+1}, \ldots, B_n$ in the same way. Therefore, we have

$$\mathcal{I}', \mu \not\models \mathsf{Neg}(H_1) \wedge \ldots \wedge \mathsf{Neg}(H_k) \wedge B_{l+1} \wedge \ldots \wedge B_n.$$

This implies $\mathcal{I}', \mu \models C$ which is a contradiction to the assumption $\mathcal{I}', \mu \not\models C$.

(iii) C was created by (4.3) or (4.4) of Definition 4.2.10 and therefore C has the form

$$\bot \leftarrow \mathsf{Neg}(A) \wedge A.$$

Furthermore, we assume $\mathcal{I}, \mu \models C$ and $\mathcal{I}', \mu \not\models C$. Therefore $\mathcal{I}', \mu \models \mathsf{Neg}(A)$ and $\mathcal{I}', \mu \models A$. According to the construction of $\mathcal{I}'$, $\mathcal{I}$ and $\mathcal{I}'$ interpret all *Neg*-atoms in the same way. Leading to

$$\mathcal{I}, \mu \models \mathsf{Neg}(A). \tag{4.15}$$

From $\mathcal{I}, \mu \models C$ follows together with (4.15), $\mathcal{I}, \mu \not\models A$. However according to the way $\mathcal{I}'$ is constructed from $\mathcal{I}$, this means $\mathcal{I}', \mu \not\models A$. Which is a contradiction to $\mathcal{I}', \mu \models A$.

□

In the worst case the $\mathcal{K}^*$-transformation results in a clause set which is 9 times larger than the original set of DL-clauses: Consider a knowledge base $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ and with the ABox containing b assertions and k different individuals. W.l.o.g. we assume that

the ABox only contains assertions of the form $D(a)$ and the RBox is empty. Hence the size of the ABox is $|\mathcal{A}| = 2b$. Furthermore, we assume that the TBox consists of the single clause

$$C = B_1 \wedge \ldots \wedge B_j \rightarrow H_1 \vee \ldots \vee H_i$$

and the symbols of all atoms occurring in C are concepts. Hence, the size of the TBox is $|\mathcal{T}| = i + j$. The set of DL-clauses for the ABox and the TBox has the size $|\mathcal{T}| + |\mathcal{A}| = i + j + 2b$. Renaming results in the following set of DL-clauses $\mathcal{K}^*$:

$$\begin{aligned}
\{B_1 \wedge \ldots \wedge B_j &\rightarrow H_1 \vee \ldots \vee H_i,\\
\mathsf{Neg}(H_1) \wedge \ldots \wedge \mathsf{Neg}(H_i) &\rightarrow \mathsf{Neg}(B_1) \vee \ldots \vee \mathsf{Neg}(B_j),\\
\sigma(H_1)(x) \wedge \mathsf{Neg}(\sigma(H_1))(x) &\rightarrow \bot,\\
&\vdots\\
\sigma(H_i)(x) \wedge \mathsf{Neg}(\sigma(H_i))(x) &\rightarrow \bot,\\
\sigma(B_1)(x) \wedge \mathsf{Neg}(\sigma(B_1))(x) &\rightarrow \bot,\\
&\vdots\\
\sigma(B_j)(x) \wedge \mathsf{Neg}(\sigma(B_j))(x) &\rightarrow \bot,\\
\top &\rightarrow dom(a_1, \ldots, a_i)\}
\end{aligned}$$

The first clause corresponds to the original clause from the TBox. Its size is $|\mathcal{T}|$. The second clause is created by renaming and has size $|\mathcal{T}|$. Then $|\mathcal{T}|$ clauses of size 7 follow. At the end of the clause set a clause of size $i + 3$ follows. All in all the resulting set of clauses has the size

$$|\mathcal{T}| + |\mathcal{T}| + 7 \cdot |\mathcal{T}| + i + 3 \leq 9 \cdot |\mathcal{T}| + i + 3 \leq 9 \cdot |\mathcal{T}| + b + 3$$

which is eight times larger than the size of the original set of DL-clauses.

4.2.3 Using the $\mathcal{K}^*$-Transformation for ABox Evolution

The $\mathcal{K}^*$-transformation can be used to compute the different operations of ABox Evolution which were introduced in Section 4.2.1. In Section 4.2.3.1, we show how to use the $\mathcal{K}^*$-transformation to compute minimal ABox deletions. Section 4.2.3.2 illustrates that using the $\mathcal{K}^*$-transformation, minimal ABox repair can be seen as a special case of minimal ABox deletion and shows how to compute minimal ABox repair. Section 4.2.3.3 addresses the task of minimal insertion of ABox assertions and shows how to use the $\mathcal{K}^*$-transformation to compute minimal insertions.

4.2.3.1 Deletion

The operation for ABox evolution that is addressed in this section is the deletion of assertions. Given a knowledge base $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ with consistent $\mathcal{R} \cup \mathcal{T}$ and an ABox assertion $C(b)$ (or $R(a, b)$) which is supposed to be deleted from the ABox, the task

is to determine a minimal set of ABox assertions whose deletion prevents that $C(b)$ is contained in the deductive closure of the knowledge base. Recall that according to the Definition 4.2.8, $\mathsf{Neg}(\mathcal{A})$ is defined as $\{\mathsf{Neg}(A) \mid A \in \mathcal{A}\}$. Furthermore, according to Definition 2.4.55, a model M is a $\mathsf{Neg}(\mathcal{A})$-minimal model for a set of DL-clauses N iff M is a model for N and further there is no model M' for N with

$$M' \cap \mathsf{Neg}(\mathcal{A}) \subset M \cap \mathsf{Neg}(\mathcal{A}).$$

Next it is shown how to use $\mathsf{Neg}(\mathcal{A})$-minimal models to compute minimal instance deletions. For this, it is convenient to use a function Del which removes the *Neg*-prefix from all *Neg*-ground atoms contained in a model.

Definition 4.2.18 (Del Function)**.** *Let M be a model for a set of DL-clauses.* $\mathsf{Del}(M)$ *is defined as follows:*

$$\mathsf{Del}(M) = \{A \in \mathcal{A} \mid \mathsf{Neg}(A) \in M\}.$$

Intuitively, in the next definition, $Del(M)$ constitutes the set of ABox assertions supposed to be deleted from the ABox to obtain a minimal instance deletion.

Theorem 4.2.19. *Let $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ be a knowledge base with consistent $\mathcal{R} \cup \mathcal{T}$, S the set of atomic concepts and atomic roles occurring in $\mathcal{A}$, and D a delete request. Let M^S be a $\mathsf{Neg}(\mathcal{A})$-minimal model for $\mathcal{K}^* \cup \{\top \rightarrow \mathsf{Neg}(D)\}$. Then $\mathcal{A} \setminus \mathsf{Del}(M^S)$ is a minimal instance deletion of D from $\mathcal{A}$.*

Proof. Let $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ be a knowledge base with consistent $\mathcal{R} \cup \mathcal{T}$, S the set of atomic concepts and atomic roles occurring in $\mathcal{A}$, and D a delete request. Let M^S be a $\mathsf{Neg}(\mathcal{A})$-minimal model for $\mathcal{K}^* \cup \{\top \rightarrow \mathsf{Neg}(D)\}$. We have to show that $\mathcal{A} \setminus \mathsf{Del}(M^S)$ is a minimal instance deletion of D from $\mathcal{A}$. This proof can be done by showing the following two assertions:

1. After the deletion of $Del(M^S)$ from the ABox, the delete request is not contained in the deductive closure anymore:

$$\Xi(\mathcal{R} \cup \mathcal{T}) \cup (\mathcal{A} \setminus \mathsf{Del}(M^S)) \not\models D$$

2. Deleting $\mathsf{Del}(M^S)$ from the ABox results in a minimal deletion: There is no $Del' \subset \mathsf{Del}(M^S)$ with $\Xi(\mathcal{R} \cup \mathcal{T}) \cup (\mathcal{A} \setminus Del') \not\models D$.

From these assertions, Theorem 4.2.19 follows immediately.

As a start, it is necessary to introduce some terms: For a set of atoms P the set $\neg P$ is defined as

$$\neg P = \{\neg A \mid A \in P\}$$

Further, for an interpretation $\mathcal{I}$ and a set of atomic concepts and atomic roles Γ, $\mathcal{I}_{|\Gamma}$ denotes the restriction of $\mathcal{I}$ on ground Γ-atoms. Now we can start proving the above mentioned two statements.

1. We have to proof that $\Xi(\mathcal{R} \cup \mathcal{T}) \cup (\mathcal{A} \setminus \mathsf{Del}(M^S)) \not\models D$.

   Per assumption, M^S is a $\mathsf{Neg}(\mathcal{A})$-minimal model for $\mathcal{K}^* \cup \{\top \rightarrow \mathsf{Neg}(D)\}$. So

   $$M^S \models \mathcal{K}^* \cup \{\top \rightarrow \mathsf{Neg}(D)\}.$$

   Firstly, we construct a model $M^{S'}$ from M^S according to Lemma 4.2.17. This model is constructed such that $M^S \subseteq M^{S'}$. Further, all atoms $C(a)$ with $C \in S$ and $\mathsf{Neg}(C(a)) \notin M^S$ are added to $M^{S'}$ and all atoms $R(a,b)$ with $R \in S$ and $\mathsf{Neg}(R(a,b)) \notin M^S$ are added to $M^{S'}$:

   $$M^{S'} = M^S \cup \{A \in \mathsf{ground}(S, Ind(\mathcal{A})) \mid \mathsf{Neg}(A) \notin M^S\}$$

   Note that this construction ensures that there is no atom $C(a)$ with $C \in S$ where neither $C(a)$ nor $\mathsf{Neg}(C(a))$ is in $M^{S'}$ and furthermore there is no atom $R(a,b)$ with $R \in S$ where neither $R(a,b)$ nor $\mathsf{Neg}(R(a,b))$ is in $M^{S'}$. The fact that $M^S \models \mathcal{K}^* \cup \{\top \rightarrow \mathsf{Neg}(D)\}$ and model $M^{S'}$ is constructed according to Lemma 4.2.17 implies that $M^{S'}$ is not only a model for $R_S(\Xi(\mathcal{R} \cup \mathcal{T}))$ but also for $\mathcal{K}^* \cup \{\top \rightarrow \mathsf{Neg}(D)\}$:

   $$M^{S'} \models \mathcal{K}^* \cup \{\top \rightarrow \mathsf{Neg}(D)\} \tag{4.16}$$

   Next we construct a model M from $M^{S'}$ according to Proposition 4.2.11 and show that

   $$M \models \Xi(\mathcal{R} \cup \mathcal{T}) \cup (\mathcal{A} \setminus \mathsf{Del}(M^S)) \cup \{D \rightarrow \bot\}.$$

   The satisfiability of

   $$\Xi(\mathcal{R} \cup \mathcal{T}) \cup (\mathcal{A} \setminus \mathsf{Del}(M^S)) \cup \{D \rightarrow \bot\}$$

   implies 1. Since

   $$\Xi(\mathcal{R} \cup \mathcal{T}) \subseteq \mathsf{R}_S(\Xi(\mathcal{R} \cup \mathcal{T})) \subseteq \mathcal{K}^*,$$

   and according to (4.16) $M^{S'} \models \mathcal{K}^*$, it follows $M^{S'} \models \Xi(\mathcal{R} \cup \mathcal{T})$. According to Proposition 4.2.11, the interpretation of all concepts and roles occurring in $\Xi(\mathcal{T} \cup \mathcal{R})$ coincides for $M^{S'}$ and M. This implies $M \models \Xi(\mathcal{R} \cup \mathcal{T})$. Further, according to (4.16), $M^{S'} \models \{\top \rightarrow \mathsf{Neg}(D)\}$. Using Proposition 4.2.11, this implies $M \not\models D$ and therefore $M \models \{D \rightarrow \bot\}$. It remains to show, that $M \models \mathcal{A} \setminus \mathsf{Del}(M^S)$.

   $$\begin{aligned}
   \mathcal{A} \setminus \mathsf{Del}(M^S) &= \{A \in \mathcal{A} \mid A \notin \mathsf{Del}(M^S)\} \\
   &= \{A \in \mathcal{A} \mid A \notin \{A \in \mathcal{A} \mid \mathsf{Neg}(A) \in M^S\}\} \\
   &= \{A \in \mathcal{A} \mid A \notin \{A \in \mathcal{A} \mid A \notin M^{S'}\}\} \\
   &= \{A \in \mathcal{A} \mid A \notin \{A \in \mathcal{A} \mid A \notin M\}\} \\
   &= \{A \in \mathcal{A} \mid A \notin \{A \in \mathcal{A} \mid M \not\models A\}\} \\
   &= \{A \in \mathcal{A} \mid A \in \{A \in \mathcal{A} \mid M \models A\}\} \\
   &= \{A \in \mathcal{A} \mid M \models A\}
   \end{aligned}$$

Therefore $M \models \mathcal{A} \setminus \mathsf{Del}(M^S)$ leads to

$$M \models \Xi(\mathcal{R} \cup \mathcal{T}) \cup (\mathcal{A} \setminus \mathsf{Del}(M^S)) \cup \{D \to \bot\}$$

which implies 1.

2. We have to show that deleting $\mathsf{Del}(M^S)$ from the ABox results in a minimal deletion: There is no $Del' \subset \mathsf{Del}(M^S)$ with $\Xi(\mathcal{R} \cup \mathcal{T}) \cup (\mathcal{A} \setminus Del') \not\models D$.

   Per assumption M is a $\mathsf{Neg}(\mathcal{A})$-minimal model for

$$\mathcal{K}^* \cup \{\top \to \mathsf{Neg}(D)\}.$$

   We assume that there is a set Del' with $Del' \subset \mathsf{Del}(M^S)$. This means that

$$(\mathcal{A} \setminus \mathsf{Del}(M^S)) \subset (\mathcal{A} \setminus Del').$$

   So there is $A \in \mathcal{A}$ with

$$A \in \mathsf{Del}(M^S) \tag{4.17}$$

   and $A \notin Del'$. Therefore $A \notin (\mathcal{A} \setminus \mathsf{Del}(M^S))$ and $A \in (\mathcal{A} \setminus Del')$. We show that adding this one arbitrarily chosen assertion A which is contained in $(\mathcal{A} \setminus Del')$ but not in $(\mathcal{A} \setminus \mathsf{Del}(M^S))$ causes D to be contained in the deductive closure. This means that we have to show

$$\Xi(\mathcal{R} \cup \mathcal{T}) \cup (\mathcal{A} \setminus \mathsf{Del}(M^S)) \cup \{A \leftarrow\} \models D. \tag{4.18}$$

   which implies $\Xi(\mathcal{R} \cup \mathcal{T}) \cup (\mathcal{A} \setminus Del') \models D$, since $(\mathcal{A} \setminus \mathsf{Del}(M^S)) \cup \{\top \to A\} \subseteq (\mathcal{A} \setminus Del')$.

   We show (4.18) by contradiction: let us assume

$$\Xi(\mathcal{R} \cup \mathcal{T}) \cup (\mathcal{A} \setminus \mathsf{Del}(M^S)) \cup \{A \leftarrow\} \not\models D.$$

   Then there is M' with

$$M' \models \Xi(\mathcal{R} \cup \mathcal{T}) \cup (\mathcal{A} \setminus \mathsf{Del}(M^S)) \cup \{\top \to A\},$$

   and $M' \not\models D$ and therefore $M' \models \{D \to \bot\}$. This means

$$M' \models \Xi(\mathcal{R} \cup \mathcal{T}) \cup (\mathcal{A} \setminus \mathsf{Del}(M^S)) \cup \{\top \to A\} \cup \{D \to \bot\}. \tag{4.19}$$

   It follows from (4.19) that $M' \models \Xi(\mathcal{R} \cup \mathcal{T})$. According to Proposition 4.2.11 it is possible to construct M'^S such that

$$M'^S \models \mathsf{R}_S(\Xi(\mathcal{R} \cup \mathcal{T})).$$

   From the way M'^S is constructed and from $M' \models \{D \to \bot\}$ it follows

$$M'^S \models \{\top \to \mathsf{Neg}(D)\}.$$

It is easy to see, that M'^S can be extended such that $M'^S \models \mathcal{K}^*$. Therefore

$$M'^S \models \mathcal{K}^* \cup \{\top \rightarrow \mathsf{Neg}(D)\}.$$

Further, from the way M'^S is constructed from M' using Proposition 4.2.11 and $M' \models \{\top \rightarrow A\}$ follows $M'^S \models \{\top \rightarrow A\}$ and $M'^S \not\models \{\top \rightarrow \mathsf{Neg}(A)\}$.
$M' \models (\mathcal{A} \backslash \mathsf{Del}(M^S))$ together with Proposition 4.2.11 implies $M'^S \models (\mathcal{A} \backslash \mathsf{Del}(M^S))$. Intuitively this can be read as: the resulting ABox after the deletion of D obtained from model M'^S is a superset of the deletion obtained from model M^S. Therefore

$$M'^S \not\models \mathsf{Neg}(\mathcal{A} \setminus \mathsf{Del}(M^S)).$$

This means, there is no $A \in \mathcal{A}$ with $A \in \mathsf{Del}(M'^S)$ and $A \notin \mathsf{Del}(M^S)$ which means

$$\mathsf{Del}(M'^S) \subseteq \mathsf{Del}(M^S).$$

Further, $M' \models \{\top \rightarrow A\}$ together with Proposition 4.2.11 used to construct M'^S implies $M'^S \models \{\top \rightarrow A\}$ and $M'^S \not\models \{\mathsf{Neg}(A) \leftarrow\}$ and therefore $A \notin \mathsf{Del}(M'^S)$. However according to (4.17) we have $A \in \mathsf{Del}(M^S)$, which leads to

$$\mathsf{Del}(M'^S) \subset \mathsf{Del}(M^S)$$

which is a contradiction to the $\mathsf{Neg}(\mathcal{A})$ minimality of M^S and therefore M' cannot exist. So

$$\Xi(\mathcal{R} \cup \mathcal{T}) \cup (\mathcal{A} \setminus \mathsf{Del}(M^S)) \cup \{\top \rightarrow A\} \cup \{D \rightarrow \bot\}$$

is unsatisfiable which implies

$$\Xi(\mathcal{R} \cup \mathcal{T}) \cup (\mathcal{A} \setminus \mathsf{Del}(M^S)) \cup \{\top \rightarrow A\} \models D.$$

□

Example 4.2.20. *The task is to delete $D(a)$ from the DL-clauses given in Example 4.2.14. For this, the clause*

$$\{\top \rightarrow NegD(a)\}$$

is added to the result of the $\mathcal{K}^*$ transformation given in Example 4.2.14. We only give the relevant part of a $\mathsf{Neg}(\mathcal{A})$ minimal model for this set of clauses:

$$M = \{NegD(a), NegB(a), NegB(b), NegC(a), \ldots\ldots\}$$

$Del(M)$ is given as:

$$\mathsf{Del}(M) = \{D(a), B(a)\}$$

This model gives us the minimal deletion:

$$\mathcal{A}' = \{C(b), R(b,b), R(a,a)\}$$

Note that Theorem 4.2.19 can further be used for minimal deletion of a delete request D which belongs to the deductive closure of the knowledge base but is not explicitly stated in the ABox $\mathcal{A}$. Meaning that $D \notin \mathcal{A}$ but $\mathcal{T} \cup \mathcal{R} \cup \mathcal{A} \models D$. In this case it only has to be ensured that $\sigma(D)$ is in the set S mentioned in Theorem 4.2.19. This can be accomplished by manually adding $\sigma(D)$ to the set S.

Next we consider a special case of deletion. For a knowledge base $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$, Theorem 4.2.19 can only be used to construct a minimal instance deletion of D from $\mathcal{A}$ if $\mathcal{K}^* \cup \{\top \rightarrow \mathsf{Neg}(D)\}$ is satisfiable. However if $\mathcal{K}^* \cup \{\top \rightarrow \mathsf{Neg}(D)\}$ is not satisfiable, there is no $\mathsf{Neg}(\mathcal{A})$-minimal model for $\mathcal{K}^* \cup \{\top \rightarrow \mathsf{Neg}(D)\}$ and therefore we cannot use Theorem 4.2.19 for the construction of a minimal instance deletion.

Example 4.2.21. *Let $\mathcal{T}$ be a TBox containing the assertion*

$$\top \sqsubseteq C$$

stating that everything belongs to the concept C. This corresponds to the DL-clause

$$\top \rightarrow C(x).$$

Let us further consider the ABox

$$\mathcal{A} = \{C(a), B(a), C(b), B(b)\}.$$

The $\mathcal{K}^*$-transformation results in the following set of clauses $\mathcal{K}^*$

$$\begin{aligned} \{\top &\rightarrow C(x) \\ NegC(x) &\rightarrow \bot, \\ C(x) \wedge NegC(x) &\rightarrow \bot, \\ \top &\rightarrow dom(a,b)\}. \end{aligned}$$

If $C(a)$ is supposed to be deleted from $\mathcal{A}$, according to Theorem 4.2.19, it is necessary to construct $\mathsf{Neg}(\mathcal{A})$-minimal models for $\mathcal{K}^* \cup \{\top \rightarrow NegC(a)\}$. However $\mathcal{K}^* \cup \{\top \rightarrow NegC(a)\}$ is unsatisfiable. Hence it is not possible to construct a minimal instance deletion of $C(a)$ from $\mathcal{A}$ using Theorem 4.2.19. Taking a closer look at the TBox reveals the problem: the TBox claims, that everything has to belong to the concept C. So the only way to remove $C(a)$ from $\mathcal{A}$ is to remove individual a entirely from the ABox.

The next Theorem uses this idea and states how to construct minimal ABox deletions in the case that $\mathcal{K}^* \cup \{\top \rightarrow \mathsf{Neg}(D)\}$ is unsatisfiable. Please note that the requirement of $\mathcal{R} \cup \mathcal{T} \cup \mathcal{A}$ being consistent in the next theorem is not a limitation since it is always possible to repair an ABox which is inconsistent with respect to its RBox and TBox using Corollary 4.2.24.

Theorem 4.2.22. *Let $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ be a knowledge base with $\mathcal{R} \cup \mathcal{T} \cup \mathcal{A}$ consistent. Let further S be the set of atomic concepts and roles occurring in $\mathcal{A}$ and let D be a delete request of the form $D = C(a)$. If $\mathcal{K}^* \cup \{\top \rightarrow \mathsf{Neg}(D)\}$ is unsatisfiable, then $\mathcal{A}' \subseteq \mathcal{A}$ is a minimal instance deletion of D from $\mathcal{A}$, where $\mathcal{A}'$ is obtained from $\mathcal{A}$ by removing all ABox assertions containing individual a.*

Proof. Let $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ be a knowledge base with $\mathcal{R} \cup \mathcal{T} \cup \mathcal{A}$ consistent, S be the set of atomic concepts and roles occurring in $\mathcal{A}$ and $D = C(a)$ be a delete request. We have to show that in case of unsatisfiability of $\mathcal{K}^* \cup \{\top \rightarrow \mathsf{Neg}(D)\}$, $\mathcal{A}' \subseteq \mathcal{A}$ is a minimal instance deletion of D from $\mathcal{A}$, where $\mathcal{A}'$ is obtained from $\mathcal{A}$ by removing all ABox assertions containing individual a. Therefore we assume $\mathcal{K}^* \cup \{\top \rightarrow \mathsf{Neg}(D)\}$ to be unsatisfiable.

We prove that

1. $\mathcal{A}'$ is a deletion of D, meaning that $\Xi(\mathcal{R} \cup \mathcal{T}) \cup \mathcal{A}' \not\models D$.
2. $\mathcal{A}'$ is a minimal deletion of D, meaning that there is no $\mathcal{A}''$ with $\mathcal{A}' \subset \mathcal{A}'' \subseteq \mathcal{A}$ and $\Xi(\mathcal{R} \cup \mathcal{T}) \cup \mathcal{A}'' \not\models D$.

From this the Theorem immediately follows.

1. Firstly, we prove that $\Xi(\mathcal{R} \cup \mathcal{T}) \cup \mathcal{A}' \not\models D$. Since no assertion in $\mathcal{A}'$ contains individual a, $\Xi(\mathcal{R} \cup \mathcal{T}) \cup \mathcal{A}'$ has to be unsatisfiable for $\Xi(\mathcal{R} \cup \mathcal{T}) \cup \mathcal{A}' \models D$ to be true. However we claimed $\Xi(\mathcal{T}) \cup \mathcal{A}$ to be consistent and since $\mathcal{A}' \subseteq \mathcal{A}$, it follows that $\Xi(\mathcal{R} \cup \mathcal{T}) \cup \mathcal{A}'$ is consistent as well. Therefore statement 1. is true.

2. Next, we prove $\mathcal{A}'$ is a minimal deletion of D: Let us assume that $\mathcal{A}'$ is not a minimal deletion, meaning that there is $\mathcal{A}''$ with $\mathcal{A}' \subset \mathcal{A}'' \subseteq \mathcal{A}$ and $\Xi(\mathcal{R} \cup \mathcal{T}) \cup \mathcal{A}'' \not\models D$. Since $\mathcal{A}'$ is obtained from $\mathcal{A}$ by removing all ABox assertions containing individual a, A'' has to contain at least one assertion A from $\mathcal{A}$ which contains individual a. We show that
$$\Xi(\mathcal{R} \cup \mathcal{T}) \cup \mathcal{A}' \cup \{\top \rightarrow A\} \models D. \tag{4.20}$$
From this, together with $A \in \mathcal{A}''$ and $\mathcal{A}' \subseteq \mathcal{A}''$, it follows immediately that
$$\Xi(\mathcal{R} \cup \mathcal{T}) \cup \mathcal{A}'' \models D.$$
And this implies 2.).

   We show (4.20) by contradiction: Let us assume $\Xi(\mathcal{T}) \cup \mathcal{A}' \cup \{\top \rightarrow A\} \not\models D$. This means, that there is a model M with
$$M \models \Xi(\mathcal{R} \cup \mathcal{T}) \cup \mathcal{A}' \cup \{\top \rightarrow A\} \cup \{D \rightarrow \bot\}.$$
Transforming
$$\mathcal{K}' = (\mathcal{R} \cup \mathcal{T}, \mathcal{A}' \cup \{\top \rightarrow A\})$$
according to Definition 4.2.13 with S the set of concept/role symbols occurring in $\mathcal{A}$ leads to $\mathcal{K}'^*$. Because of $\mathcal{A}' \cup \{\top \rightarrow A\} \subseteq \mathcal{A}$, the only difference between $\mathcal{K}'^*$ and $\mathcal{K}^*$ is that the clause of the form $dom(a_i, \ldots, a_n)$ may contain additional individuals in $\mathcal{K}^*$. According to the model construction used in the proof of Proposition 4.2.15, we can construct M^S from M with $M^S \models \mathcal{K}'^*$. It is easy to see, that we can extend M^S to a model M' such that it is a model for $\mathcal{K}^*$ as well. Since $M \models \{D \rightarrow \bot\}$, according to Proposition 4.2.11, $M^S \models \mathsf{Neg}(D)$ and $M' \models \mathsf{Neg}(D)$. Hence
$$M' \models \mathcal{K}^* \cup \{\top \rightarrow \mathsf{Neg}(D)\}.$$

This is a contradiction to the unsatisfiability of $\mathcal{K}^* \cup \{\top \rightarrow \mathsf{Neg}(D)\}$ leading to (4.20).

□

With the help of Theorem 4.2.19 and 4.2.22, for delete requests of the form $D = C(a)$, we are now able to construct minimal instance level deletions independent from the satisfiability of

$$\mathcal{K}^* \cup \{\top \rightarrow \mathsf{Neg}(D)\}.$$

Next, repair of an ABox which is inconsistent w.r.t. its TBox and RBox is considered. In order to compute minimal repair, the introduced minimal deletion turns out to be helpful.

4.2.3.2 Repair

This section addresses the repair operation for ABoxes as introduced in Section 4.2.1.2. Given a knowledge base $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ with consistent $\mathcal{R} \cup \mathcal{T}$ and inconsistent $\mathcal{T} \cup \mathcal{R} \cup \mathcal{A}$, the task is to find a minimal set of ABox assertions whose deletion causes $\mathcal{A}$ to be consistent w.r.t. $\mathcal{T} \cup \mathcal{R}$. The $\mathcal{K}^*$-transformation introduced in Definition 4.2.13 can be used to repair an ABox, which is inconsistent w.r.t. its TBox and RBox. The basic idea is to replace each occurrence of $\bot$ in $\mathcal{T}$ by a new atom *false* and further add *false* to the set S from Theorem 4.2.19. After that, we use the $\mathcal{K}^*$-transformation and construct the minimal instance deletion of *false* from the ABox. The resulting ABox is a minimal ABox repair.

Lemma 4.2.23. *Let $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ be a knowledge base with consistent $\mathcal{R} \cup \mathcal{T}$, $\mathcal{T}_{false}$ the TBox obtained from $\mathcal{T}$ by replacing every occurrence of $\bot$ by false, $\mathcal{A}_{false}$ be $\mathcal{A} \cup \{false\}$ and $\mathcal{K}_{false} = (\mathcal{R}, \mathcal{T}_{false}, \mathcal{A}_{false})$. Let S be the set of atomic concepts and roles occurring in $\mathcal{A}$ and $\mathcal{T}$ plus false. Then there is a $\mathsf{Neg}(\mathcal{A})$-minimal model for $\mathcal{K}^*_{false} \cup \{\top \rightarrow Negfalse\}$.*

Proof. Let $\mathcal{K}_{false} = (\mathcal{R}, \mathcal{T}_{false}, \mathcal{A}_{false})$ be defined as in Lemma 4.2.23 and S be the set of atomic concepts and roles occurring in $\mathcal{A}$ and $\mathcal{T}$ plus *false*. We show the satisfiability of

$$\mathcal{K}^*_{false} \cup \{\top \rightarrow Negfalse\}.$$

Since $\mathcal{T}$ is consistent, *false* does not follow from $\Xi(\mathcal{T}_{false})$. Therefore $\Xi(\mathcal{T}_{false}) \cup \{false \rightarrow \bot\}$ is satisfiable. From Proposition 4.2.15 it follows, that K'^* is satisfiable, with

$$K' = (\mathcal{R} \cup \mathcal{T}_{false} \cup \{false \rightarrow \bot\}, \mathcal{A}_{false}).$$

Transforming $\mathcal{K}'$ according to Definition 4.2.13 leads to:

$$\mathcal{K}'^* = \mathcal{K}^*_{false} \cup \{false \rightarrow \bot, \top \rightarrow Negfalse, false \wedge Negfalse \rightarrow \bot\}.$$

It is easy to see, that

$$\mathcal{K}^*_{false} \cup \{\top \rightarrow Negfalse\} \subset \mathcal{K}'^*.$$

Therefore $\mathcal{K}^*_{false} \cup \{\top \rightarrow Negfalse\}$ has to be satisfiable. This implies, that there has to be a $\mathsf{Neg}(\mathcal{A})$-minimal model for $\mathcal{K}^*_{false} \cup \{\top \rightarrow Negfalse\}$. □

Corollary 4.2.24. *Let $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ be a knowledge base with consistent $\mathcal{R} \cup \mathcal{T}$, $\mathcal{T}_{false}$ the TBox obtained from $\mathcal{T}$ by replacing every occurrence of $\bot$ by false, $\mathcal{A}_{false}$ be $\mathcal{A} \cup \{false\}$ and $\mathcal{K}_{false} = (\mathcal{R}, \mathcal{T}_{false}, \mathcal{A}_{false})$. Let S be the set of atomic concepts and roles occurring in $\mathcal{A}$ and $\mathcal{T}$ plus false. Then $\mathcal{A} \setminus Del(M)$ is a minimal ABox repair for $\mathcal{A}$ for all $\mathsf{Neg}(\mathcal{A})$-minimal models M for $\mathcal{K}^*_{false} \cup \{\top \rightarrow Negfalse\}$.*

Corollary 4.2.24 follows immediately from Theorem 4.2.19 with $D = false$. Lemma 4.2.23 together with Corollary 4.2.24 implies that such a minimal ABox repair can always be constructed.

4.2.3.3 Insertion

Next we address insertion of an assertion into an existing ABox. Given a knowledge base $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$, the task is to insert an assertion like $C(a)$ into the ABox such that the resulting ABox is consistent w.r.t. the TBox and the RBox. This can be obtained, by first adding the assertion to the ABox and afterwards constructing all possible minimal repairs for the resulting ABox. If the added assertion is not contained in any of these minimal ABox repairs, then it is not possible to insert the assertion into the ABox without rendering the ABox inconsistent w.r.t. its TBox. If there is a minimal repair containing the added assertion, then the insertion is possible and the respective minimal ABox repair gives us the result of the insertion.

Example 4.2.25. *Reconsider the knowledge base $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ with an empty RBox given in Example 4.2.5. The TBox is represented by the following DL-clause*

$$\Xi(\mathcal{T}) = \{\bot \leftarrow C(x) \wedge D(x)\}.$$

Furthermore, the ABox is given as

$$\mathcal{A} = \{C(a)\}.$$

Adding the assertion $D(a)$ into $\mathcal{A}$ leads to

$$\mathcal{A} = \{C(a), D(a)\}$$

which is inconsistent w.r.t. $\mathcal{T}$. In order to be able to insert $D(a)$ into the ABox anyway, we can repair $\mathcal{A}'$. There are two minimal ABox repairs for $\mathcal{A}'$: $\mathcal{A}'' = \{C(a)\}$ and $\mathcal{A}''' = \{D(a)\}$. The first minimal repair corresponds to deleting the previously inserted $D(a)$ and therefore is not desirable. The second minimal repair however allows us to keep the inserted assertion.

At a first glance, insertion requires the construction of all possible repairs. However it is possible to find out if an insertion is possible by a single satisfiability test: If we want to add $C(a)$ to the ABox, we first add it to $\mathcal{A}$ and then add the following DL-clause

$$NegC(a) \rightarrow \bot$$

to the knowledge base as well. If the resulting knowledge base is unsatisfiable, we can conclude that it is not possible to add $C(a)$ to $\mathcal{A}$ such that the resulting knowledge base is satisfiable. If the resulting knowledge base is satisfiable, we get a minimal repair as minimal model and know that the insertion can be successfully performed.

As illustrated in the previous sections, the $\mathcal{K}^*$-transformation can be used to compute minimal deletion, minimal insertion and minimal repair. In all three cases, only the deviation from the original ABox has to be computed. In practice it is reasonable to assume that usually only a very small part of an ontology has to be revised when computing deletion, insertion and repair. This is why it seems likely that the $\mathcal{K}^*$-transformation provides an efficient method for the computation of ABox evolution. In order to check this assertion, a prototypical implementation of the described methods for ABox evolution based on the $\mathcal{K}^*$-transformation was developed. In Section 4.2.4 experimental results are presented.

4.2.4 Evaluation

In order to test the practicality of using the $\mathcal{K}^*$-transformation for ABox deletion, insertion and repair, a prototypical implementation of a system[1] as well as some of the experimental results were developed as a Master's thesis (Polster, 2015). for ABox evolution was developed (Polster, 2015). The implementation uses the Hyper theorem prover (Baumgartner et al., 2007) to construct the $\mathsf{Neg}(\mathcal{A})$-minimal models which lead to the minimal deletions. Another theorem prover able to handle DL-clauses is HermiT (Motik et al., 2007). However, in contrast to Hyper, HermiT is not able to compute $\mathsf{Neg}(\mathcal{A})$-minimal models. This is why the Hyper theorem prover is better suited for our purposes.

The main goal of the experiments described in this section is to test the scalability of using the $\mathcal{K}^*$-transformation for ABox deletion, insertion and repair. This is why the experiments are designed to see how the performance of the $\mathcal{K}^*$-transformation for ABox deletion, insertion and repair depends on the number of assertions in the ABox.

To the best of our knowledge, there is no system performing deletion and insertion of ABox assertions as well as repair of inconsistent ABoxes for the description logic $\mathcal{SHI}$. This is why we cannot compare our system to another system.

Section 4.2.4.1 introduces some optimizations that can be carried out before the computation of the $\mathcal{K}^*$-transformation. Section 4.2.4.2 gives details on how to compute $\mathsf{Neg}(\mathcal{A})$-minimal models using the Hyper theorem prover. Section 4.2.4.3 introduces the setup of our experiments and in Section 4.2.4.4 experimental results for using the $\mathcal{K}^*$-transformation for ABox deletion, insertion and repair are presented.

4.2.4.1 Optimizations Included in the Implementation

The approach for minimal deletion, insertion and repair introduced in Section 4.2.3.1 is based on the assumption that most parts of the ABox do not have to be changed in order to perform the change operation. For example when performing a deletion,

[1] Implementation as well as ontologies used are available at `http://userp.uni-koblenz.de/~obermaie/ABoxEvolution.zip` [2015, December 17]

only a very small fraction of the ABox has to be removed for the deletion to come into effect. Therefore, the number of assertions contained in the ABox should not have a wide influence on the performance of our implementation. Nevertheless, for implementation purposes, it is advantageous not to consider more ABox assertions then necessary. This is why the ABox is partitioned in advance. This partitioning can be seen as a recompilation technique since it has to be performed only once. The partitioning of ABox assertions will be defined in terms of an equivalence relation which is induced by an equivalence relation on the set of individuals occurring in the ABox. Recall that $\mathsf{Ind}(\mathcal{A})$ is the set of individuals occurring in $\mathcal{A}$.

Definition 4.2.26 ($\sim_{\mathsf{Ind}(\mathcal{A})}$ Relation on ABox Individuals)**.** *Let $\mathcal{A}$ be an ABox. The relation $\sim_{\mathsf{Ind}(\mathcal{A})} \subseteq \mathsf{Ind}(\mathcal{A}) \times \mathsf{Ind}(\mathcal{A})$ is given as follows:*

$$a \sim_{\mathsf{Ind}(\mathcal{A})} b \text{ iff there is a concept } C \text{ with } \{C(a), C(b)\} \subseteq \mathcal{A} \text{ or } R(a,b) \in \mathcal{A}.$$

It is easy to see that the $\sim_{\mathsf{Ind}(\mathcal{A})}$-relation is an equivalence relation. The $\sim_{\mathsf{Ind}(\mathcal{A})}$-relation on ABox individuals induces an equivalence relation on ABox assertions as follows:

Definition 4.2.27 ($\sim_{\mathcal{A}}$ Relation on ABox Assertions)**.** *Let $\mathcal{A} = \{A_1, \ldots, A_n\}$ be an ABox. The relation $\sim_{\mathcal{A}} \subseteq \mathcal{A} \times \mathcal{A}$ is defined as follows:*

$$A_i \sim_{\mathcal{A}} A_j \text{ iff there is an } a_1 \in \mathsf{Ind}(A_i) \text{ and an } a_2 \in \mathsf{Ind}(A_j) \text{ with } a_1 \sim_{\mathsf{Ind}(\mathcal{A})} a_2.$$

Like in the case of the $\sim_{\mathsf{Ind}(\mathcal{A})}$-relation for individuals, it is easy to see that the extension of $\sim_{\mathcal{A}}$ on ABox assertions is an equivalence relation. The $\sim_{\mathcal{A}}$ equivalence relation can be used to partition the ABox before computing the $\mathcal{K}^*$-transformation. Without partitioning, the $\mathcal{K}^*$-transformation adds the DL-clause $\{\top \rightarrow dom(a_1, \ldots, a_n)\}$ for all individuals occurring in $\mathcal{A}$. The basic idea of partitioning is to add only those individuals into this DL-clause which are necessary to answer a delete request. Given a delete request D only those individuals equivalent to the ones occurring in D have to be added to the DL-clause $\{\top \rightarrow dom(a_1, \ldots, a_n)\}$. If large ABoxes with many individuals are considered, this can be helpful in order to relieve the theorem prover. Since the computation of insertion and repair relies on deletion, this technique is helpful for the computation of insertion and repair as well.

4.2.4.2 Using Hyper to Compute Γ-minimal Models

In Section 4.2.2 we briefly discussed the complexity of the $\mathcal{K}^*$-transformation. There is a linear blow up of the knowledge base and there is also polynomial time complexity for performing the transformation. The real costs for performing the deletion, insertion and repair are caused by the theorem prover which has to compute the $\mathsf{Neg}(\mathcal{A})$-minimal models. Dix, Furbach, and Niemelä (2001) provides an overview on this topic. In our implementation we use the Hyper theorem prover for the construction of $\mathsf{Neg}(\mathcal{A})$-minimal models. Hyper is able to construct Γ-minimal models in a bottom-up way. For this a set of DL-clauses together with a set of predicate symbols P and an integer i is handed to Hyper. Then Hyper only constructs models containing at most i ground instances constructed

from predicate symbols from P. During reasoning, Hyper discards all branches with more than i ground instances constructed form predicate symbols from P. If Hyper is not able to find a model with i or less such ground instances, it terminates by stating that the maximal number of instances is reached.

However this Hyper feature cannot be used directly to compute $\mathsf{Neg}(\mathcal{A})$-minimal models. We illustrate this fact with an example.

Example 4.2.28. *Consider the following ABox*

$$\mathcal{A} = \{A(a), A(b), B(c)\}$$

$\mathsf{Neg}(\mathcal{A})$ *corresponds to the set* $\{\mathsf{Neg}A(a), \mathsf{Neg}A(b), \mathsf{Neg}B(c)\}$. *Telling Hyper to construct models containing at most one ground instance constructed from predicate symbols in* $\{\mathsf{Neg}A, \mathsf{Neg}B\}$ *causes Hyper to abandon branches containing* $\mathsf{Neg}A(c)$ *and* $\mathsf{Neg}B(a)$ *even though this branch could still lead to a* $\mathsf{Neg}(\mathcal{A})$*-minimal model. The reason for this is the fact that Hyper is not able to distinguish between those ground instances occurring in* $\mathsf{Neg}(\mathcal{A})$ *and those not occurring in* $\mathsf{Neg}(\mathcal{A})$. *We solve this problem by introducing two auxiliary functions renaming concepts and roles.*

Definition 4.2.29 (delete Function, ABox Function)**.** *Let* $A(a)$ *and* $R(a, b)$ *atoms. The* delete *and the* ABox *function map atoms to renamed atoms as follows:*

- *For atomic concepts* A *and an individual or variable* a*:*
  - $\mathsf{delete}(A(a)) = deleteA(a)$
  - $\mathsf{ABox}(A(a)) = ABoxA(a)$
- *For atomic roles* R *and individuals or variables* a*,* b*:*
  - $\mathsf{delete}(R(a, b)) = deleteR(a, b)$
  - $\mathsf{ABox}(R(a, b)) = ABoxR(a, b)$

We slightly abuse notation by using the delete function to rename atomic concepts and atomic roles: for B an atomic concept or a role: $\mathsf{delete}(B) = deleteB$. Hence the delete function to rename atoms, sets of atoms and atomic concepts and roles.

Now we present present how Hyper can be used to compute $\mathsf{Neg}(\mathcal{A})$-minimal models: As mentioned before, the DL-clause

$$\top \rightarrow dom(a_1, \ldots, a_n)$$

with $\{a_1, \ldots, a_n\} = Ind(\mathcal{A})$ is added by $\mathcal{K}^*$-transformation. The main reason for this is to introduce the individuals occurring in the ABox to the Hyper theorem prover. For implementation purposes, we add a different set of DL-clauses to introduce those

No.	Hardware	OS
1	AMD Phenom X6 1090T @ 3.2GHz, 8 GB RAM	Debian Wheezy
2	Intel Core i5-2410M @ 2.3GHz, 8 GB RAM	Ubuntu 14.04
3	Intel(R) Core(TM)2 Quad CPU Q9550 @ 2.83GHz, 4 GB RAM	Ubuntu 14.04

Table 4.1: Different Setups Used for the Experiments.

individuals. Instead of adding the aforementioned DL-clause,

$$\{\top \rightarrow \mathsf{ABox}(A) \mid \text{for all assertions } A \in \mathcal{A}\}$$
$$\cup$$
$$\{\mathsf{ABox}(A)(x) \wedge \mathsf{Neg}(A)(x) \rightarrow \mathsf{delete}(A)(x) \mid A(a) \in \mathcal{A} \text{ for an individual } a\}$$
$$\cup$$
$$\{\mathsf{ABox}(R)(x,y) \wedge \mathsf{Neg}(R)(x,y) \rightarrow \mathsf{delete}(R)(x,y) \mid R(a,b) \in \mathcal{A} \text{ for individuals } a,b\}$$

is added to the result of the $\mathcal{K}^*$-transformation. By telling Hyper to only construct models containing at most i ground instances of delete predicates, it is ensured that Hyper only considers those Neg predicates for minimization, for which there is a corresponding ground instance in the ABox. This avoids the problem illustrated in Example 4.2.28.

The construction of $\mathsf{Neg}(\mathcal{A})$-minimal models is done by iteratively calling Hyper. Firstly, Hyper is called with the resulting set of DL-clauses and the instruction to construct models containing at most one ground instance of delete predicates. The maximal number of ground instances of delete predicates allowed is successively increased until Hyper is either able to construct a model or a proof for the unsatisfiability of the set of DL-clauses. This ensures that the first model given by Hyper is a $\mathsf{Neg}(\mathcal{A})$-minimal model.

4.2.4.3 Setup of Experiments

We used three different setups for our experiments. Table 4.1 provides an overview on these setups. The main goal of our experiments is to observe the influence of the number of ABox assertions on the runtime of our implementation. Therefore it is not harmful that we used different setups for the experiments since we did not change the setup during one test series. For all experiments the Hyper version 1.0[2] was used.

We use the $\mathcal{ALHI}$ ontology VICODI[3] for testing our approach. This is a manually created ontology for the domain of European history. The VICODI ontology comes in

[2] Available at: `http://userpages.uni-koblenz.de/~bpelzer/hyper/` [2015, December 17].
[3] http://www.vicodi.org [2015, December 17].

five different sizes. All versions of this ontology consists of 223 axioms in the TBox and RBox. The smallest version contains 53653 ABox assertions. The larger versions of this ontology are generated by duplicating the assertions of the original ABox several times and changing the names of the individuals in the assertions. The smallest version of the VICODI ontology is denoted by VICODI_0, VICODI_n denotes the ontology obtained by copying the ABox n times and then changing the names of the individuals. With its 223 axioms, the TBox of the VICODI ontologies is small and the structure is simple. In contrast to that, the ABox is large and contains many different individuals which are connected via roles.

Since our implementation uses the partitioning described in Section 4.2.4.1, using the larger versions of the VICODI ontology for the experiments does not provide a benefit because of their repetitive structure. This is why, to evaluate the scalability of the $\mathcal{K}^*$-transformation for ABox deletion, insertion and repair, we focus on the smallest version of the VICODI ontology by constructing different versions of it with increasing numbers of ABox assertions. These fragments are used as test sets throughout the experiments. Six versions of the smallest VICODI ontology were considered. The largest version corresponds to VICODI_0, the smaller versions are created by successively removing 10,000 assertions from the ABox. In the remainder of this section, we refer to this set of ontologies as *fragments of VICODI_0*. The implementation[4] as well as some of the experimental results were developed as a Master's thesis (Polster, 2015).

4.2.4.4 Experimental Results

For each of the three change operations, namely deletion, insertion and repair, experiments were performed. All experiments are performed using the fragments of the VICODI_0 ontology. Whenever we present runtimes in this Section, these times contain everything from loading the ontology, partitioning the ontology, performing the $\mathcal{K}^*$-transformation and the respective change operation including using Hyper to compute the $\mathsf{Neg}(\mathcal{A})$-minimal models. For deletion, different assertion occurring in the ABox of the ontology under consideration were treated as delete request. For repair, firstly random inconsistencies were introduced into the ABox which then were removed using the $\mathcal{K}^*$-transformation. For insertion, random assertions were created and inserted into the ABox such that the resulting ABox is consistent w.r.t. its TBox.

Deletion Before we describe the experiments performed for deletion, we introduce different kinds of deletions: If it is sufficient to only remove the delete request itself from the ABox in order to minimally delete the delete request, we speak of *atomic deletions*. In contrast to this *non-atomic deletion* denote cases were it is necessary to remove more than one ABox assertion. Furthermore, note that in all experiments on deleting assertions, only one minimal instance deletion is computed since this is sufficient to perform the deletion.

[4] Implementation as well as ontologies used are available at `http://userp.uni-koblenz.de/~obermaie/ABoxEvolution.zip` [2015, December 17]

1. The first experiments on deletion compare the runtime of our implementation of the $\mathcal{K}^*$-transformation to the runtime of Hyper without the $\mathcal{K}^*$-transformation to perform the same deletion. Hyper alone can be used to compute minimal deletions as well. The basic idea to use Hyper alone to handle a delete request D is to delete a set N of assertions from the ABox $\mathcal{A}$ and then test if D is contained in the deductive closure of $\mathcal{T} \cup \mathcal{A} \setminus N$. For non-atomic deletions this approach is not feasible since all possible subsets N of the ABox have to be considered and it has to be ensured that the deleted set of assertions is minimal. This is why we use this approach only for atomic deletions. This can be done as follows: If the task is to test if D can be removed from the ABox by removing only D from the ontology KB, we can test

$$KB \setminus \{D\} \cup \{\neg D\}$$

for satisfiability using Hyper. Satisfiability of $KB \setminus \{D\} \cup \{\neg D\}$ implies

$$KB \setminus \{D\} \not\models D,$$

meaning that D can be deleted atomically.

For each of the fragments of VICODI_0 described above, our system used 1,000 different ABox assertions as delete request D and performed the deletion. Furthermore, we used Hyper to compute the same deletions. Note that only our implementation uses the $\mathcal{K}^*$-transformation. This experiment was carried out using setup one given in Table 4.1.

The solid line in Figure 4.3 presents the average time our system used for a delete request leading to an atomic deletion on the different fragments of VICODI_0. The dashed line in Figure 4.3 presents the time used for those atomic deletions computed by Hyper without the $\mathcal{K}^*$-Transformation.

Comparing the lines for Hyper and our system using the $\mathcal{K}^*$-transformation shows that the $\mathcal{K}^*$-Transformation is faster in calculating atomic deletions. It is notable, that our system first implements the $\mathcal{K}^*$-transformation and then uses Hyper to compute a $\mathsf{Neg}(\mathcal{A})$-minimal model. Nevertheless Figure 4.3 shows that our system outperforms Hyper alone. This can be explained by the fact that our implementation only computes the deviation from the ABox and especially in the case of atomic deletions, this deviation is very small. In contrast to this, Hyper is forced to compute the deductive closure of the ABox which contains much more information than necessary.

We did not perform experiments comparing our system to Hyper without $\mathcal{K}^*$-transformation for non-atomic deletions. As described afore, it is not feasible to perform non-atomic deletions with Hyper alone. Since it is reasonable to expect our system to outperform Hyper without $\mathcal{K}^*$-transformation for non-atomic deletions, we refrain form such experiments.

2. The next experiments analyze the scalability of atomic and non-atomic deletions dependend on the number of assertions in the ABox. They were carried using setup

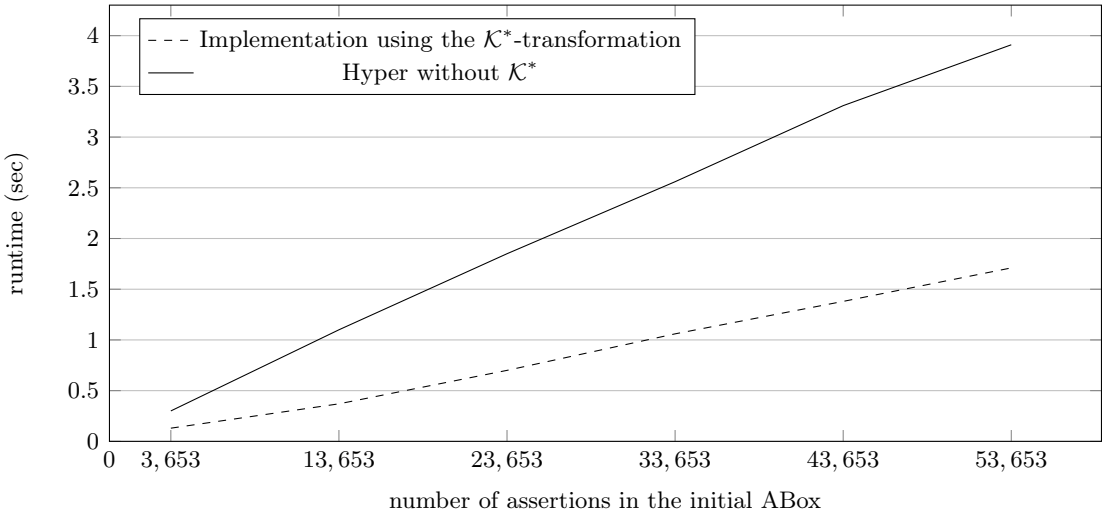


Figure 4.3: Average Time Used for Deleting one Assertion from the Different Fragments of the VICODI_0 Ontology. Average Time Computed from 1,000 Different Deletions. Only Atomic Deletions are Considered.

two given in Table 4.1. During these experiments, for each fragment ontology of VICODI_0, each assertion in the ABox was treated as a delete request in turn. Figure 4.4 depicts runtimes of our implementation of the $\mathcal{K}^*$-transformation for atomic as well as non-atomic deletions. The solid line presents the runtimes for atomic deletions and the dashed line the runtimes for the non-atomic deletions. For both atomic and non-atomic deletions, Figure 4.4 reveals a nice property of the $\mathcal{K}^*$-transformation: increasing the size of the ABox only leads to a harmless increase of the time necessary to compute the minimal deletion. We owe this property to the fact that only the deviation from the original ABox is computed. As described in Section 4.2.4.2, for the calculation of non-atomic deletions it is necessary to restart Hyper's model generation process. This implies that the runtime of non-atomic deletions depends greatly on the number of ABox assertions that have to be deleted in order to accomplish the delete request. This is also the reason, why the non-atomic deletions in Fiure 4.4 take longer than atomic deletions. These experiments revealed another interesting aspect, which is not visible in Figure 4.4: In most cases of non-atomic deletions, it was only necessary to delete two or three assertions. This observation confirms that our assumption that deletion of an assertion usually only affects a small part of the ABox applies for these experiments. Furthermore, it explains that the computation time of a non-atomic deletion only increases moderately when the size of the ABox under consideration is increased.

The next experiments analyze repair of an ABox which is inconsistent w.r.t. its TBox and RBox.

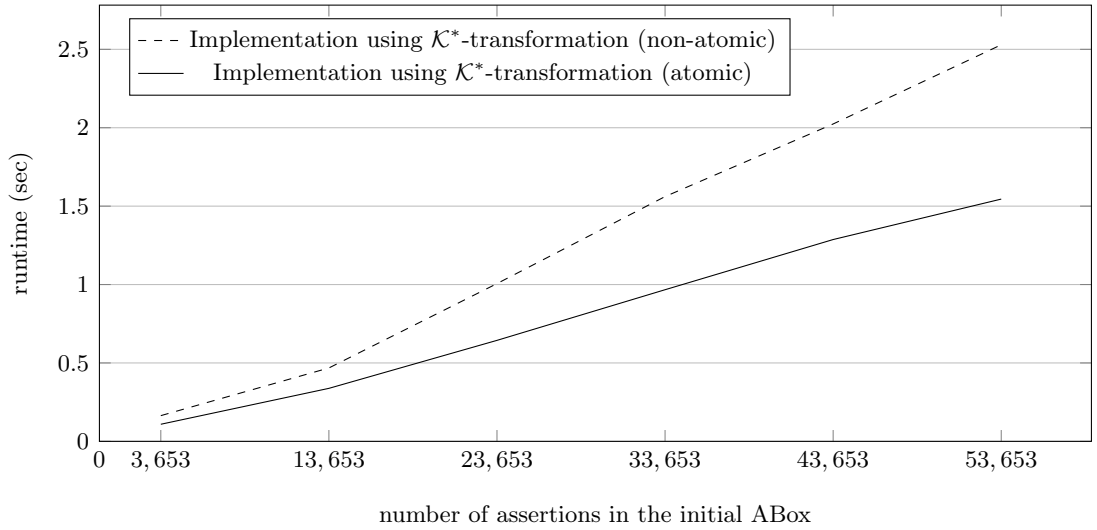


Figure 4.4: Average Time Used for Deleting one Assertion from the Different Fragments of the VICODI_0 Ontology. Average Time Computed from Using each ABox Assertion as a Delete Request in Turn.

Repair All fragments of the VICODI_0 ontology contain an ABox which is consistent w.r.t. the rest of the knowledge base. Therefore, in order to perform experiments for ABox repair, it is necessary to first introduce inconsistencies into the ontology. Our approach only addresses the aspect of an ABox which is inconsistent w.r.t. its TBox and RBox. Therefore, both the TBox and the RBox remain fixed during the generation of inconsistencies and only the ABox $\mathcal{A}$ is changed. For the generation of inconsistencies, an approach similar to the one introduced by Bienvenu, Bourgaux, and Goasdoué (2014) was used. All changes are performed depending on a given probability p:

- For all assertions $C(a)$ occurring in $\mathcal{A}$, $C(a)$ is replaced by $\neg C(a)$ with probability p. Since the ABox is assumed to contain only assertions of the form $C(a)$ or $R(a, b)$ with C an atomic concept, adding $\neg C(a)$ to the ABox is done by adding $C'(a)$ to the ABox for a fresh atomic concept C' and furthermore adding the DL-clause $C(x) \wedge C'(x) \rightarrow \bot$ to the TBox.
- For all role assertions $R(a, b)$ occurring in $\mathcal{A}$, $R(a, b)$ is replaced by $\neg R(a, b)$ with probability $p/10$. Again the introduction of $\neg R(a, b)$ is done by introducing $R'(a.b)$ for a fresh role R' into the ABox and furthermore adding the DL-clause $R(x, y) \wedge R'(x, y) \rightarrow \bot$ to the TBox.
- For all role assertions $R(a, b)$ occurring in $\mathcal{A}$, $R(a, b)$ is replaced by $R(b, a)$ with probability $p/10$.

The approach for ABox evolution presented in the thesis at hand is based on the assumption that usually only a very small part of the ABox has to be removed in order to repair an ABox which is inconsistent w.r.t. its TBox and RBox. This is why the probability p

was set to the rather small value of 0.01%. This probability results in changing up to 15 assertions to the ABoxes considered for our experiments. To avoid that in each case the changes are returned as a minimal repair, it is claimed that the introduced changes are preserved in the repair.

1. In the first experiments, the procedure described afore was used to create 1,000 inconsistent ABoxes for each fragment of vidodi_0. Figure 4.5 shows the average number of changed assertions using the method described above with $p = 0.01\%$ for the different VICODI_0 fragments. For the changes introduced into the ABoxes of the different fragments of VICODI_0, minimal repairs were computed with the help of the $\mathcal{K}^*$-transformation. For technical reasons, only one minimal repair was computed but all *smallest* minimal repairs. The smallest minimal repairs are exactly those minimal repairs, with a minimal size. These experiments were carried out using setup three given in Table 4.1. The results of this computation depicted in Figure 4.6 show that there is a moderate increase of runtime for growing number of ABox assertions. We assume that this increase is not mainly inflicted by the growing number of assertions in the ABox under consideration. Contrariwise, we assume that the main reason for the increasing runtime is the higher number of assertions which have to be deleted in order to resolve the introduced inconsistency. To support this theory, another set of experiments was conducted investigating the runtime depending on the number of assertions which have to be deleted to repair the knowledge base.

2. The next experiments were performed as a follow up of the first experiments on ABox repair. They are designed to investigate the relationship between runtime and the number of assertions which have to be deleted to repair the knowledge base under consideration. This relationship is more insightful that the relationship between runtime and number of changes caused by the introduction of inconsistencies since even very small changes to an ABox can result in a very high number of ABox assertions which have to be deleted in order to regain consistency. For example our experiments on the smallest fragment of the VICODI_0 ontology revealed that changing only one assertion of the ABox can lead to up to 564 assertions which have to be removed in order to reestablish consistency. Figure 4.7a depicts runtime dependent on the number of assertions which have to be deleted to regain consistency for the smallest fragment of the VICODI_0 ontology.

   These experiments were carried out for all fragments of the VICODI_0 ontology. See Figure 4.7 and Figure 4.8 for the results. The experiments reveals two things: Firstly, the larger the part of the ABox which has to be removed, the higher the runtime of our implementation. Secondly, the runtime is not much influenced by the size of the ABox itself. These tests were carried using setup 3 given in Table 4.1.

   Figure 4.9 summarizes the results depicted in Figure 4.7 and Figure 4.8. and shows the runtime dependent on the number of assertions which have to be deleted for all fragments of the VICODI_0 ontologies. Comparing the graphs for the

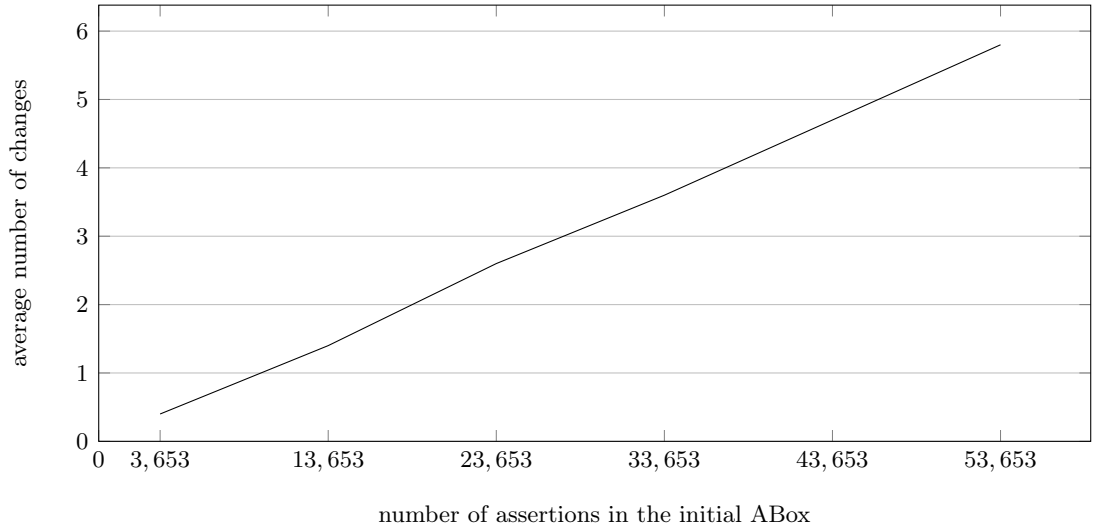


Figure 4.5: Average Number of Assertions Changed to Make the Fragments of the VICODI_0 Ontology Inconsistent with $p = 0.01\%$.

different number of assertions shows that the runtime increases only moderately for increasing number of ABox assertions undermining our thesis that runtime primarily depends on the number of assertions which have to be deleted in order to regain consistency.

Insertion The third operation for ABox evolution introduced in Section 4.2.1.3 is the insertion of an assertion into an ABox. The resulting ABox should to be consistent w.r.t. its TBox and RBox and furthermore it has to be ensured that inserted assertion is contained in the resulting ABox. As described in Section 4.2.1.3, the changes to the ABox performed to achieve this have to be minimal. The next experiments are designed to investigate how the runtime of the computation of an insertion is influenced by the number of assertions in an ABox. Our implementation uses the $\mathcal{K}^*$-transformation to insert an assertion into an ABox as described in Section 4.2.3.3. For this, an assertion of the form $C(a)$ or $R(a, b)$ is created randomly. The probability of the assertion to be a role assertion is equal to the probability of the assertion to be a concept assertion. The assertion is created by randomly picking a concept or a role together with one or two individuals occurring in the knowledge base. The created assertion is added to the ABox and then repair is used while ensuring that the inserted assertion stays in the ABox. For each ontology under consideration, 1,000 different assertions were randomly created and inserted. These experiments were carried out using setup three given in Table 4.1.

Figure 4.10 depicts the runtimes for the different fragments of the VICODI_0 ontology. It is noteworthy that the runtime only increases moderately for growing ABox sizes. In most cases, the ABox obtained by adding the randomly created insertion was consistent. Hence no repair was necessary. Insertion heavily relies on repair. Therefore, runtimes for cases where the ABox resulting from adding the assertion is inconsistent, depends on

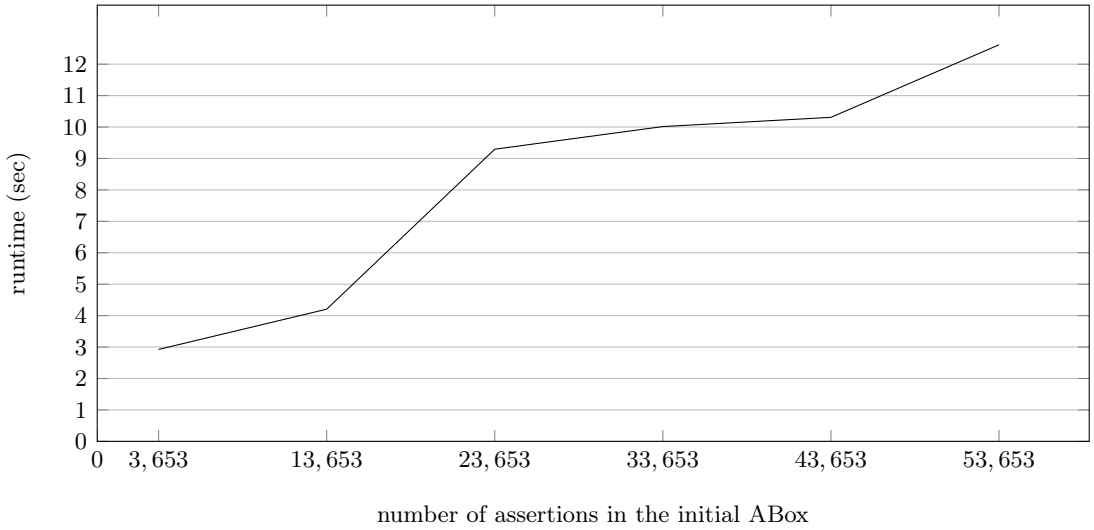


Figure 4.6: Runtime of Repair for the Different Fragments of the VICODI_0 Ontology with Introduced Inconsistencies Using $p = 0.01\%$.

the number of assertions which have to be deleted in order to regain consistency.

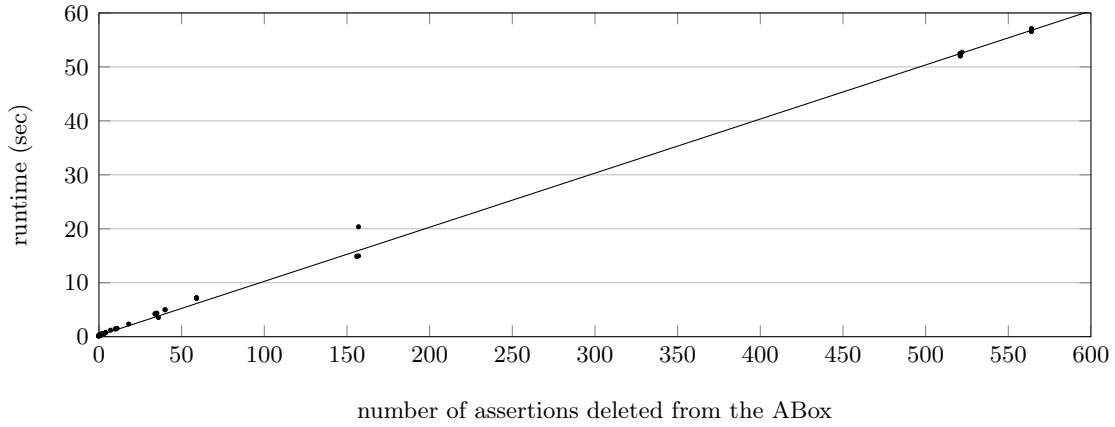
To sum up, all experiments carried out, confirm our assertion that the runtime depends mainly on the dimension of change which is inflicted on the ABox and is only moderately influenced by the size of the ABox itself. This makes our approach beneficial for the computation of minimal deletion, insertion and repair for ABoxes where only a rather small part of the ABox is affected by the change operation. Next we consider approaches, which are related to the method for ABox evolution we presented.

4.3 Related Work

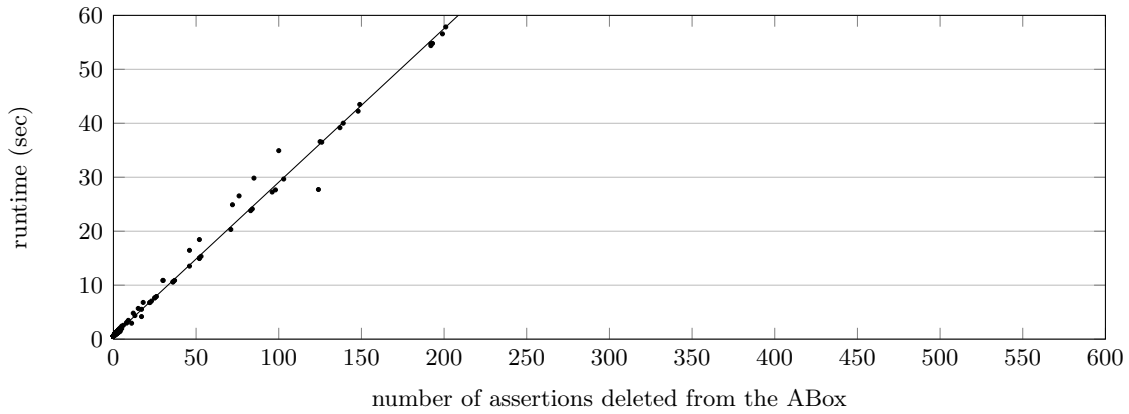
Naturally, ontologies used in practice are subject to frequent changes. These changes concern different parts of an ontology. Coarsely, these changes can be divided into changes affecting the terminological part i.e. the TBox and changes concerning the assertional part i.e. the ABox of an ontology. There is a plethora of approaches to deal with this changes. This thesis only examines ABox updates. Hence we will not present details on TBox updates and refer the reader to (Zheleznyakov, Calvanese, Kharlamov, and Nutt, 2010).

For the task of updating an ABox, the TBox of the given knowledge base is considered to stay fixed. In contrast to this, the ABox is subject to changes. In general, two kind of ABox updates can be distinguished. The *formula based* and the *model based* approach. In both approaches changes carried out are supposed to fulfill some minimality criterion.

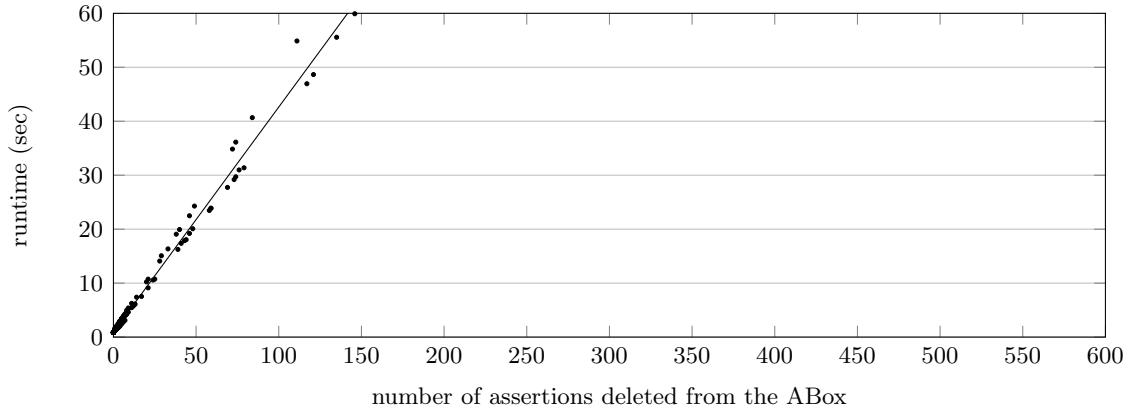
For the two approaches, this minimality criterion has a different meaning. In case of the formula based approach, this view of minimal change concerns the assertions of the ABox itself meaning that the number of assertions changed by the update are supposed



(a) Fragment of VICODI_0 Ontology with 3,653 Assertions.

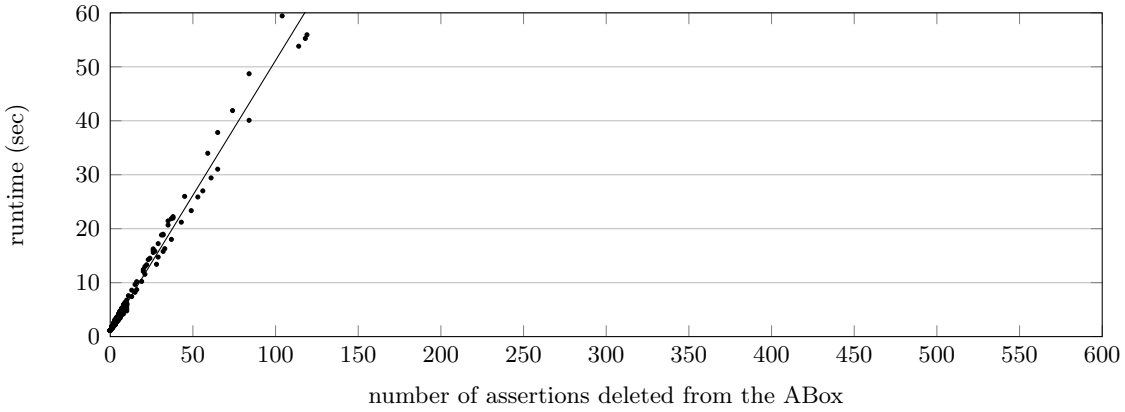


(b) Fragment of VICODI_0 Ontology with 13,653 Assertions.

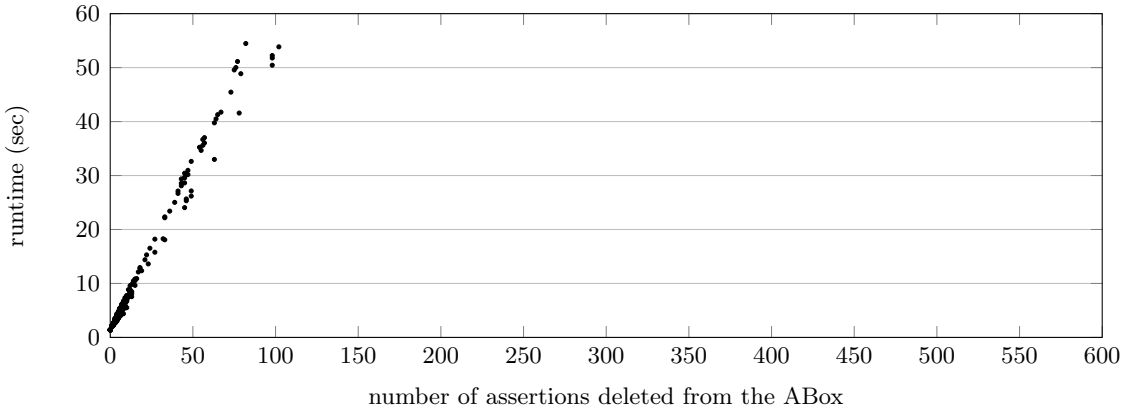


(c) Fragment of VICODI_0 Ontology with 23,653 Assertions.

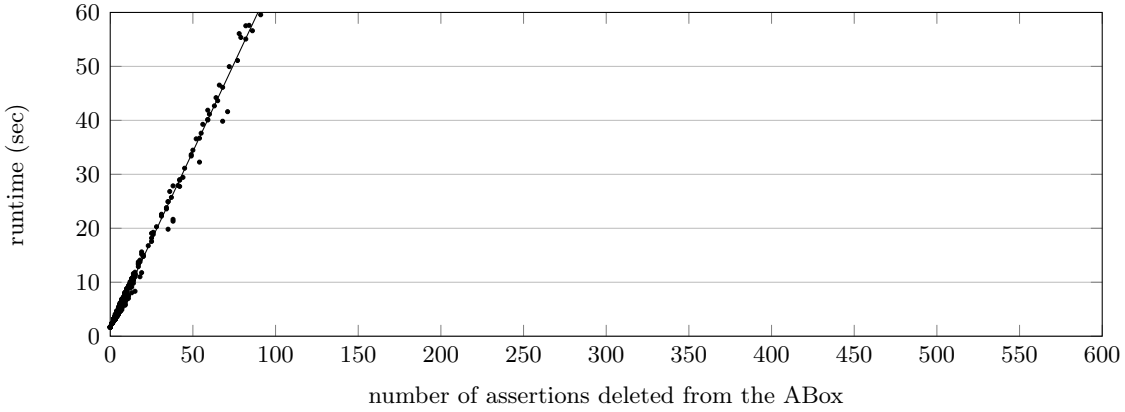
Figure 4.7: Runtimes of Repair Dependent on the Number of Assertions which were Deleted (represented by the dots in the graph) for the Three Smallest Fragments of the VICODI_0 Ontology with Introduced Inconsistencies Using $p = 0.01\%$.



(a) Fragment of VICODI_0 Ontology with 33,653 Assertions.



(b) Fragment of VICODI_0 Ontology with 43,653 Assertions.



(c) Whole VICODI_0 Ontology with 53,653 Assertions.

Figure 4.8: Runtimes of Repair Dependent on the Number of Causes which were Deleted for the Three Largest Fragments of the VICODI_0 Ontology with Introduced Inconsistencies Using $p = 0.01\%$.

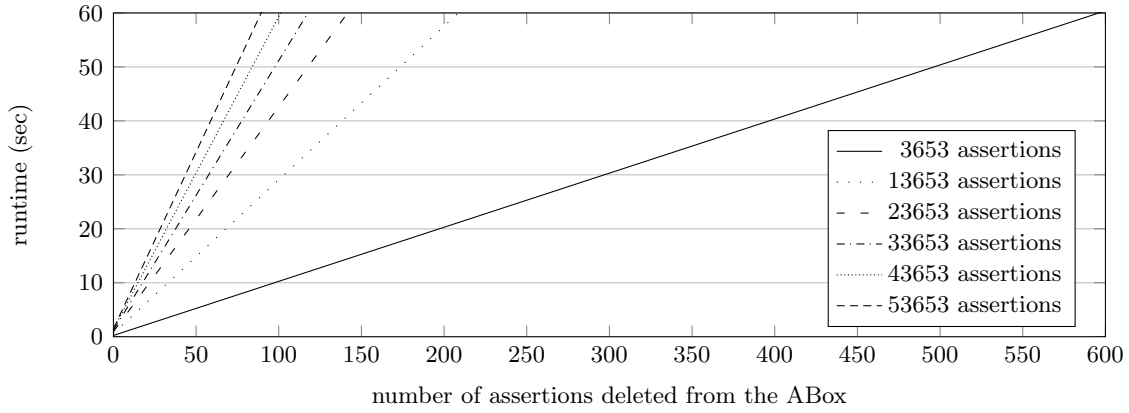


Figure 4.9: Average Runtimes of Repair Dependent on the Number of Assertions which were Deleted for the Different Fragments of the VICODI_0 Ontology with Introduced Inconsistencies Using $p = 0.01\%$.

to be minimal. The semantically guided evolution of $\mathcal{SHI}$ ABoxes introduced in this dissertation belongs to this category of ABox updates.

In case of the model based approach, the minimality criterion refers to the models of the knowledge base. The model based approach claims that the set of models of the knowledge base resulting from a change operation should be as close as possible to the set of models of the original knowledge base (Liu, Lutz, Milicic, and Wolter, 2006). In Section 4.3.1 we briefly introduce the idea of model-based ABox updates together with some results on which description logics allow model-based ABox updates. In Section 4.3.2 some approaches for formula based ABox updates are introduced. Section 4.3.2.2 introduces the notions of axiom pinpointing and justifications, together with methods how to compute them.

Section 4.3.3 introduces some ideas from querying large ABoxes. Due to the size of the considered ABoxes, it is likely that the ABoxes under consideration are inconsistent w.r.t. their TBox and RBox. Therefore, the field of querying large ABoxes has to deal with inconsistent ABoxes as well.

4.3.1 Model Based ABox Updates

In order to introduce the idea of model based ABox updates, we first have to introduce some terms. In Section 2.4.1 the description logic $\mathcal{ALC}$ was presented. Extending this logic with inverse roles leads to the description logic $\mathcal{ALCI}$. Furthermore, it is possible to add so-called qualified number restrictions to $\mathcal{ALC}$ allowing to construct concepts of the form$\geq nR.C$ or $\leq nR.C$ respectively. This can be formalize for example a father with at least three sons as $\mathit{Father} \sqcap \geq 3\mathit{hasChild}.\mathit{Male}$. Semantics of these qualified number

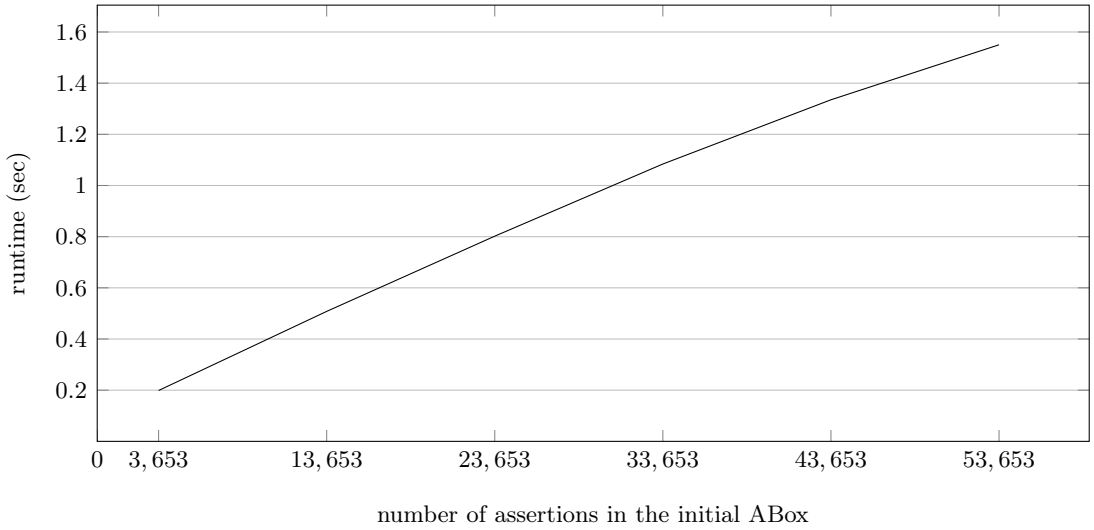


Figure 4.10: Runtimes for Insertion of an Assertion into the Different Fragments of the VICODI_0 Ontology.

restrictions is given by extending Definition 2.4.8 with the following two lines

$$(\geq n\, R.C)^{\mathcal{I}} = \{x \mid |\{y \mid (x,y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}| \geq n\}$$
$$(\leq n\, R.C)^{\mathcal{I}} = \{x \mid |\{y \mid (x,y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}| \leq n\}.$$

Adding qualified number restrictions to $\mathcal{ALC}$ results in the description logic $\mathcal{ALCQ}$. So called nominals allow the use of individual names in the description of a concept. For an individual name a, $\{a\}$ is a concept. This concept is interpreted as the set consisting only of the individual a. For example it is possible to formalize the concept of actors who have worked with Quentin Tarantino as $Actor \sqcap \exists worked\_with.\{tarantino\}$. Adding nominals to $\mathcal{ALC}$ results in the description logic $\mathcal{ALCO}$. By *description logics between* $\mathcal{ALC}$ *and* $\mathcal{ALCQI}$ all logics obtained by adding qualified number restrictions or invers roles to $\mathcal{ALC}$.

With the help of the '@' constructor, it is possible to state that an individual belongs to a certain concept. Given an individual a and a concept C, it is possible to write $@_a C$ which is interpreted as $\Delta^{\mathcal{I}}$, if $a^{\mathcal{I}} \in C^{\mathcal{I}}$ and as $\emptyset$ otherwise. For example it is possible to formalize that Quentin Tarantino is a screenwriter and film director as $@_{tarantino}(Screenwriter \sqcap Director)$.

With the help of the above introduced notions we are now able to introduce the task of model-based ABox updates. When performing model-based ABox updates, the models of the knowledge base resulting from the update should be as close as possible to the models of the original knowledge base. Liu et al. (2011) introduces this kind of updates as *semantic updates*. An ABox $\mathcal{A}'$ is a semantic update of an ABox $\mathcal{A}$ with a set of assertions $\mathcal{U}$ if the models of $\mathcal{A}'$ are the interpretations obtained from the models of $\mathcal{A}$ by making all assertions in $\mathcal{U}$ true. However, according to Liu et al. (2011) for the

description logics between $\mathcal{ALC}$ and $\mathcal{ALCQI}$ it is not always possible to express these semantic updates. If nominals and the '@' constructor from hybrid logic are added to these logics, semantic updates can be expressed. To give an intuition, why it is necessary to have nominals in order to be able to express semantic updates, we present an example presented by Liu et al. (2011).

Example 4.3.1. *(Liu et al., 2011)*

$$\begin{aligned}\mathcal{A} \quad = \{ \quad & (Person \sqcap \exists hasChild.Person \sqcap \forall hasChild.(Person \sqcap Happy))(john), \\ & hasChild(john, peter), \\ & Person(mary)\}\end{aligned}$$

Description logic ABoxes represent incomplete knowledge about the world, which means that the semantics of ABoxes can be characterized as open world semantics. In the ABox $\mathcal{A}$ it is not clear if the individual Mary belongs to the concept Happy or not. Further it is not stated if Mary is John's child. Now the ABox is updated such that Mary is unhappy. So the set $\mathcal{U}$ is added to the ABox with:

$$\mathcal{U} = \{(\neg Happy)(mary)\}$$

Performing a semantic update, leads to the following ABox:

$$\begin{aligned}\mathcal{A}' \quad = \{ \quad & (Person \sqcap \exists hasChild.Person \sqcap \\ & \forall hasChild.(Person \sqcap (Happy \sqcup \{mary\})))(john), \\ & hasChild(john, peter), \\ & (Person \sqcap \neg Happy)(mary)\}\end{aligned}$$

Since it cannot be excluded that Mary is John's child in the ABox $\mathcal{A}$, it is necessary to change the description of John by adding the nominal $\{mary\}$.

Since the thesis at hand introduces an approach for formula based ABox updates, we refrain from presenting more details on model based approaches. See (Liu et al., 2011) and more detailed (Liu, 2010) for different notions of model based ABox and information on how to compute some of these notions and (De Giacomo, Lenzerini, Poggi, and Rosati, 2009) for an approach for the description logic DL-Lite.

4.3.2 Formula Based ABox Updates

The basic idea of formula based ABox updates is to minimally change the ABox to perform the update. In this setting, the minimality of change means that as little as possible assertions of the ABox are changed. As opposed to the model based approach, there are no restrictions on the models of the knowledge base.

Besides the approach for formula based ABox updates for $\mathcal{SHI}$ knowledge bases presented in this thesis and Furbach and Schon (2013a), there are numerous other formula based approaches to deal with the evolution of the instance level of knowledge bases given in

different description logics. Many of the other approaches focus on description logics belonging to the so-called DL-Lite family (Calvanese, Giacomo, Lembo, Lenzerini, and Rosati, 2007). To understand the idea of the approaches that are introduced throughout this section, it is not important to know details about DL-Lite. This is why we refrain from presenting DL-Lite and refer the reader to the respective paper cited along with each approach for details on the logic under consideration.

One interesting approach to compute formula based ABox updates is presented by Halaschek-Wiener et al. (2006), where so-called syntactic ABox updates are introduced. Similar to our approach, assertions are added to or removed from the ABox. In the case of deleting ABox assertions, however it is neither guaranteed that the removed assertion is not contained in the deductive closure nor that the result of adding the assertion is consistent.

Calvanese, Kharlamov, Nutt, and Zheleznyakov (2010) introduce a formula based approach for ABox deletion, insertion and repair for DL-Lite knowledge bases. This approach exploits the following nice property of DL-Lite: as shown by Calvanese et al. (2010), in DL-Lite the unsatisfiability of an ABox w.r.t. a TBox is either caused by a single assertion or a pair of assertions. However in $\mathcal{SHI}$ an arbitrary number of assertions can cause unsatisfiability w.r.t. a TBox. Therefore, the algorithms suggested by Calvanese et al. (2010) are not feasible for $\mathcal{SHI}$ knowledge bases.

In Section 4.3.2.1 inconsistency-tolerant semantics are introduced which are based on the notion of minimal repair as introduced in Section 4.2.1.2. Furthermore, it is shown how to answer queries form an ABox which is inconsistent w.r.t. its TBox using these inconsistency-tolerant semantics. Section 4.3.2.2 presents how to use justifications to compute formula based ABox updates together with an idea how to compute justifications. We end the presentation of formula based approaches for ABox updates with the introduction of a method to compute maximal consistent subsets of an ontology in Section 4.3.2.3.

4.3.2.1 Inconsistency-Tolerant Semantics

Querying an inconsistent ABox in the classical way is useless since it is possible to deduce anything from an inconsistent ABox. This thesis pursues the idea to repair an ABox which is inconsistent w.r.t the TBox. In general there can be more than one way to repair an inconsistent ABox and it is unclear which of these possible ways leads to the desired ABox. Furthermore, it is not necessarily the case that there is consensus on what the desired ABox is. The thesis at hand provides a method to efficiently compute minimal ABox repairs. However deciding which of these repairs should be used is beyond the scope of this thesis.

Another possibility to deal with these inconsistencies is to introduce inconsistency tolerant semantics (Lembo, Lenzerini, Rosati, Ruzzi, and Savo, 2010). These semantics provide a way to answer queries in a meaningful way even though the ABox is inconsistent. There are many different ways to define inconsistency tolerant semantics. Usually those semantics use the notion of a repair as given in Definition 4.2.3. Lembo et al. (2010) establishes inconsistency-tolerant semantics in order to be able to use those inconsistent

ABoxes for query answering. Rosati (2011) studies the complexity of reasoning under these inconsistent-tolerant semantics. To use these inconsistent-tolerant semantics, it is necessary to determine minimal repairs of the ABox. Algorithms for the computation of minimal repair of DL-Lite ABoxes suggested in Lembo et al. (2010) test the satisfiability of every single ABox assertion and every pair of ABox assertions w.r.t. the TBox. Since for DL-Lite the satisfiability test is tractable, this approach is reasonable. However the ExpTime completeness of consistency testing of $\mathcal{SHI}$ ABoxes renders such an approach infeasible. Furthermore, the algorithms suggested in Lembo et al. (2010) cannot be used for $\mathcal{SHI}$ ABoxes because these algorithms exploit the property of DL-Lite that unsatisfiability of an ABox w.r.t. a TBox is either caused by a single assertion or a pair of assertions. In contrast to this, in $\mathcal{SHI}$ an arbitrary number of assertions can cause unsatisfiability w.r.t. a TBox.

Bienvenu and Rosati (2013b) present a summary of the most common inconsistency tolerant semantics. Since query answering under inconsistency tolerant semantics is computationally expensive, methods to approximate some of these semantics are presented as well. Recall that there can be more than one minimal repair for a knowledge base $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$. As given in Definition 4.2.3, $Rep(\mathcal{K})$ denotes the set of all possible minimal repairs. Since Bienvenu and Rosati (2013b) only consider knowledge bases given in DL-Lite there is no RBox in the following. Furthermore, the term *query* denotes a first-order logic formula without free variables and as usual, a query is entailed by a knowledge base iff it is true in all models of the knowledge base. We give the formal definition of three different types of inconsistency tolerant semantics and provide explanations to each of the semantics afterwards.

Definition 4.3.2 (Inconsistency Tolerant Semantics (Lembo et al., 2010; Bienvenu and Rosati, 2013b))**.** *Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a knowledge base. A query q is entailed by $\mathcal{K}$ under*

- consistent query answering *(CQA) semantics, written as $(\mathcal{T}, \mathcal{A}) \models_{CQA} q$ iff $(\mathcal{T}, \mathcal{B}) \models q$ for every repair $\mathcal{B} \in Rep(\mathcal{K})$,*
- intersection ABox repair *(IAR) semantics, written as $(\mathcal{T}, \mathcal{A}) \models_{IAR} q$ iff $(\mathcal{T}, \mathcal{D}) \models q$ where $\mathcal{D} = \bigcap_{\mathcal{B} \in Rep(\mathcal{K})} \mathcal{B}$,*
- brave semantics*, written as $(\mathcal{T}, \mathcal{A}) \models_{brave} q$ iff $(\mathcal{T}, \mathcal{B}) \models q$ for some repair $\mathcal{B} \in Rep(\mathcal{K})$.*

Consistent Query Answering Semantics Each repair in $Rep(\mathcal{K})$ keeps as much as possible from the original ABox. Hence assuming that one of these repairs corresponds to a correct representation of the assertional knowledge seems to be reasonable. However, it is unclear, which of the minimal repairs in $Rep(\mathcal{K})$ is the most accurate. A careful way of reasoning suggests that a query q can only be considered to be a consequence of $\mathcal{K}$ if it is a consequence of $(\mathcal{T}, \mathcal{A}')$ for all repairs $\mathcal{A}' \in Rep(\mathcal{K})$. This is the basic idea of the most prominent example for semantics dealing with querying inconsistent knowledge bases which was introduced by Lembo et al. (2010) and is called *consistent query answering semantics* (CQA-semantics). The CQA-semantics allow to deduce queries which

intuitively seem to be reasonable consequences of the knowledge base. However the drawback of these semantics is that even for description logics with very limited expressivity these semantics are intractable w.r.t. data complexity (Bienvenu, 2012; Lembo et al., 2010). The reason for this is the fact that in the worst case the number of repairs can be exponential in the size of the ABox of the knowledge base.

Intersection ABox Repair Semantics To remedy the computationally intractability of CQA semantics, the more conservative intersection ABox repair (IAR) semantics are suggested. In the case of IAR semantics, a query only follows, if it follows from the intersection of all possible minimal repairs. In some settings, query answering under IAR semantics has the same computational complexity of standard query answering. See (Lembo, Lenzerini, Rosati, Ruzzi, and Savo, 2011) for details. In IAR semantics, only those assertions not involved in any contradictions are used. This is rather restrictive and can cause some inferences to be missed. Depending on the application, these missed inferences could be of interest. This is why, Bienvenu and Rosati (2013b) introduces *brave semantics*.

Brave Semantics In brave semantics a query follows if there is a minimal repair from which it follows. Obviously, under these semantics much more queries follow than under CQA or IAR semantics. Brave semantics however have the drawback that it is possible to deduce contradictory queries. Hence, it should rather be used to show that a certain query does not hold. When considering these CQA, IAR and brave semantics, it is obvious that it would be desirable to have a combination of CQa and IAR semantics. This is addressed by approximations of CQA semantics which will be introduced next.

Approximations of Consistent Query Semantics Bienvenu and Rosati (2013b) introduce two additional families of inconsistency-tolerant semantics which try to combine the advantages of the CQA and the IAR semantics. One of these families, the k-support semantics, form an under-approximation and the other, k-defeater semantics, an over-approximation of the CQA-semantics. Both approximations use the notion of a $\mathcal{T}$-support of a query. Given a knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, a subset S of $\mathcal{A}$ is a $\mathcal{T}$-support of a query q if S is consistent w.r.t $\mathcal{T}$ and $(\mathcal{T}, S) \models q$. A $\mathcal{T}$-support is a minimal $\mathcal{T}$-support if no proper subset of it is a $\mathcal{T}$-support.

As mentioned before, a query q is entailed under CQA semantics if q is entailed by every repair. This means that every repair contains at least one $\mathcal{T}$-support for q. Another way to put this is that there is a set $\{S_1, \ldots, S_n\}$ of $\mathcal{T}$-supports for q, such that every repair contains some S_i. The idea of the k-support semantics is to restrict the number of $\mathcal{T}$-supports which are allowed to be used to the number k.

Definition 4.3.3 (k-Support Semantics (Bienvenu and Rosati, 2013b))**.** *Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a knowledge base and $k \in \mathbb{N}$. A query q is entailed by $\mathcal{K}$ under* k-support semantics*, written as $\langle \mathcal{T}, \mathcal{A} \rangle \models_{k-supp} q$ iff there are subsets $S_1, \ldots, S_k$ of $\mathcal{A}$ with:*

- *each S_i is a $\mathcal{T}$-support for q in $\mathcal{A}$ and*

- *for every $R \in Rep(\mathcal{K})$, there is some S_i with $S_i \subseteq R$.*

Intuitively it is clear that: if q is entailed under k-support semantics, q is entailed under $k+1$-support semantics as well. By successively increasing k, it is possible to approximate CQA semantics more and more closely. However the k-support semantics can not be used to show, that a query is not entailed. This is why another approximation is introduced in Bienvenu and Rosati (2013b).

The idea of this second approximation is the following: If a query q is not entailed under CQA semantics, there is a repair $\mathcal{B} \in Rep(\mathcal{K})$ with $\langle \mathcal{T}, \mathcal{B} \rangle \not\models q$. This means, that $\mathcal{B}$ contradicts all $\mathcal{T}$-supports of q. The k-defeater semantics now work with the idea of restricting the number of facts in $\mathcal{B}$. A query is entailed under k-defeater semantics if there is no way to construct a repair $\mathcal{B}$ containing at most k facts, which contradicts all $\mathcal{T}$-supports of q.

Definition 4.3.4 (k-defeater Semantics (Bienvenu and Rosati, 2013b))**.** *Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a knowledge base and $k \in \mathbb{N}$. A query q is entailed by $\mathcal{K}$ under* k-defeater semantics*, written as $\langle \mathcal{T}, \mathcal{A} \rangle \models_{k-def} q$ iff there is no subset $\mathcal{B}$ of $\mathcal{A}$ which is consistent w.r.t. $\mathcal{T}$ with $|\mathcal{B}| \leq k$ such that $\langle \mathcal{T}, \mathcal{B} \cup C \rangle \models \bot$ for every minimal $\mathcal{T}$-support $C \subseteq \mathcal{A}$ of q.*

It is immediately clear that for every $k \geq 1$ if $\mathcal{K} \models_{k+1-def} q$ then $\mathcal{K} \models_{k-def} q$. So the k-defeater semantics can be used to over-approximate CQA-semantics closer and closer.

We conclude the section on inconsistent tolerant semantics by introducing preferred repair semantics.

Preferred Repair Semantics As mentioned before, there can be more than one repair for an inconsistent knowledge base. In general it is not clear which one of these repairs is the most desirable one or there even is no consensus what constitutes a desired repair. In some cases however, additional information about the ABox assertions is available, which can be used to determine if a repair is adequate or not. Bienvenu et al. (2014) assumes that such information is available in form of additional information on the reliability of the different ABox assertions. This information is used to find so-called preferred repairs which are used to define variants of consistent query semantics and IAR semantics. The reliability of the ABox assertions can be seen as a preorder over subsets of ABox assertions. With the help of this preorder, repairs which are minimal w.r.t. the reliability are defined. These reliability information minimal repairs are then used as a basis for variants of consistent query semantics and IAR semantics.

The next formula based approach for ABox updates is based on so-called justifications. Those justifications are very similar to $\mathcal{T}$-supports used in Definition 4.3.3 and Definition 4.3.4.

4.3.2.2 Approaches Based on Justifications

The basic idea of the approach for instance level deletion presented in this thesis can be summarized as follows: Given a knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{R}, \mathcal{A})$ and an ABox assertion A supposed to be deleted, the method determines a minimal subset of the ABox which

has to be removed from the ABox in order to prevent A to be contained in the deductive closure of the knowledge base. Another way to perform minimal instance level deletion would be to first compute all subsets of the ABox which, together with the TBox and the RBox, entail A. In the second step, a hitting set of all these subsets is constructed. Each minimal hitting set corresponds to a minimal instance deletion of A. Our approach does not perform the first step of this method. We construct the minimal hitting set directly, avoiding to construct all subsets of the ABox entailing the delete request. The task to find minimal subsets of the ABox which, together with the TBox and the RBox, entail an assertion is closely related to the calculation of so-called justifications.

For a given consequence, identifying justifications is the task to find the minimal subsets of the knowledge base under consideration, having this consequence. See (Baader and Peñaloza, 2010), (Kalyanpur, Parsia, Horridge, and Sirin, 2007) and (Horridge, 2011) for details. Given an ontology $\mathcal{K}$ and an entailment D in $\mathcal{K}$, a justification is a minimal subset of that ontology such that the entailment still holds in the subset. Note that a justification does not only contain ABox assertions but TBox assertions as well. Furthermore, D is an entailment, meaning that it has the form $C \sqsubseteq E$, with C, E concepts. So finding a minimal deletion of an ABox assertion $A(b)$ can be seen as the special case to find justification for $\top \sqsubseteq A(b)$. According to Baader, Peñaloza, and Suntisrivaraporn (2007), even for very inexpressive description logics, the number of justifications for an entailment is exponential in the size of the ontology.

Halaschek-Wiener, Katz, and Parsia (2006) addresses revision of ABoxes in the description logic $\mathcal{SHOIN}$ by exploiting the notion of justifications. The description logic $\mathcal{SHOIN}$ is obtained from the description logic $\mathcal{SHI}$ by adding nominals and so-called number restrictions. With the help of number restrictions it is possible to construct concepts of the form $\geq nR$ and $\leq nR$ making it possible to formalize concepts like a father with at least three children as $\mathit{Father} \sqcap \geq 3\mathit{hasChild}$. Semantics is given by extending the semantics of $\mathcal{SHOI}$ with

$$(\geq n\, R)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid |\{b \mid (a,b) \in R^{\mathcal{I}}\}| \geq n\},$$
$$(\leq n\, R)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid |\{b \mid (a,b) \in R^{\mathcal{I}}\}| \leq n\}.$$

Given a $\mathcal{SHOIN}$ ontology $\mathcal{K}$ and an ABox assertion D, a *justification* of D in $\mathcal{K}$ is a subset of $\mathcal{K}$, implying D. If no proper subset of a given justification for D is a justification for D as well, the justification is called *minimal justification*. The basic idea of the *revision operator* introduced by Halaschek-Wiener et al. (2006) is to first insert the new information D into the knowledge base. In the next step, all justifications for $\bot$ in the resulting knowledge base are computed and then assertions are removed from each justification. This approach ensures that the resulting knowledge base is consistent. However D is not guaranteed to be contained in the resulting knowledge base. This is why the operator corresponding to this approach is called a *semi-revision*.

Example 4.3.5. *Consider the following TBox and ABox:*

$$\mathcal{T} = \{C \sqcap D \sqsubseteq \bot\}$$
$$\mathcal{A} = \{D(a)\}.$$

Adding the assertion $C(a)$ to the ABox leads to an inconsistent knowledge base. There are two justifications for $\bot$ in the resulting knowledge base. One justification is $\{C(a)\}$ and the other is $\{D(a)\}$. One possibility would be to remove the previously inserted $C(a)$ form the ABox. The resulting knowledge base is consistent but does not contain the inserted information. Another possibility would be to remove $D(a)$ from the ABox resulting in the new ABox $\mathcal{A}' = \{C(a)\}$.

The approach of using justifications for revision is closely related to the idea of insertions introduced in Section 4.2.3.3. To insert some ABox assertion, first the assertion is added to the ABox and afterwards all possible repairs of the resulting knowledge base are constructed. However in the approach introduced in the thesis at hand, it is ensured that the inserted assertion is contained in the resulting knowledge base. Next we consider how to compute justifications.

Computing Justifications When considering approaches to construct justifications, it is first distinguished between approaches constructing a single justification and approaches constructing all justifications. In addition to that, according to Horridge (2011), approaches can be divided into black-box and glass-box algorithms.

Black-box algorithms systematically construct different subsets S of the ontology and check if $S \models D$. For the test of $S \models D$ an arbitrary reasoner can be used. This is why algorithms belonging to this category can be seen as black-box approaches. Algorithms in this category usually work in two phases. An expansion phase which constructs a subset of the ontology entailing D followed by a contraction phase in which this set is minimized. In the expansion phase, starting from a small set of assertions $S \subseteq \mathcal{K}$, $S \models D$ is checked. If $S \not\models D$, the set S is expanded and checked again. This procedure is repeated until a subset S is constructed, which entails D. This means that either S or a subset of S is a justification for D in $\mathcal{K}$. It is easy to see that it is crucial how S is expanded. This is why there are numerous optimizations for this phase. Given a subset S of the ontology, a selection function is be used to choose which axioms to add to S in one step of the expansion phase. A very simple example could be a selection function adding all axioms to S sharing symbols with S. Details on more sophisticated selection functions are presented byHorridge (2011). Another way to optimize the expansion phase is to use modularization of the ontology. The basic idea of this optimization is that the justification does not have to be searched in the whole ontology but in a module of the ontology. If the module is guaranteed to preserve the entailment of D, this optimization can be very helpful since the module is much smaller than the whole ontology. See for example (Baader and Suntisrivaraporn, 2008) and (Grau, Horrocks, Kazakov, and Sattler, 2007). As mentioned before, at the end of the expansion phase, a subset S with $S \models D$ is reached and either S or a subset of S is a justification for D in $\mathcal{K}$. Therefore, in the next phase, the so-called contraction phase, S is gradually contracted until a minimal set is obtained, which is a justification of D in $\mathcal{K}$. In a naive approach, one assertion is removed from S in each step of the contraction phase. There are numerous optimizations for this phase. See (Kalyanpur et al., 2007) for details.

Glass-box approaches for the construction of justifications are usually closely connected

to reasoners. Therefore, it is not possible to use an arbitrary reasoner. In contrast to that, it is necessary to adjust the reasoner for the creation of justifications. These algorithms use a technique called tracing to generate justifications. The basic idea of tracing is to tag reasoning structures during the reasoning process. For example in the completion-ABox of a tableau, the different assertions are tagged by the input axioms used to generate them. Therefore, these tags can be used to construct justifications. See (Schlobach and Cornet, 2003; Lee, Meyer, Pan, and Booth, 2006; Kalyanpur, 2006) for some examples.

There are approaches for the construction of concise justifications. Horridge, Parsia, and Sattler (2008) introduces laconic and precise justifications. Roughly speaking, laconic justifications are not allowed to contain superfluous parts. However these approaches focus on the TBox of a knowledge base and therefore we do not present details here and refer to (Horridge et al., 2008).

4.3.2.3 Maximal Consistent Subontologies

Haase and Stojanovic (2005) introduce an approach to formalize the semantics of change of OWL ontologies. Three different notions of consistency are presented: structural consistency, user-defined consistency and logical consistency. In the context of the thesis at hand, only logical consistency is of interest and we refer the reader to (Haase and Stojanovic, 2005) for the other notions of consistency. The whole ontology is seen as a set of axioms and it is not distinguished between TBox, RBox and ABox. This means, that the approach presented can be used for both the evolution of an ABox and a TBox. In order to re-establish logical consistency after changing the ontology, a maximal consistent subontology is created. An algorithm for finding such a maximal consistent subontology is presented. The basic idea of this algorithm is to iteratively remove axioms from the inconsistent ontology and to test the resulting ontology for consistency. This procedure is repeated until a consistent ontology is reached. A selection function provides the axioms which have to be removed. Since, in the worst case all possible subsets of the ontology have to be tested, it is crucial to use an adequate selection function. One selection function suggested by Haase and Stojanovic (2005) works by selecting axioms based on structural connectedness, where two axioms are called structurally connected if they share a symbol. In each step, the algorithm constructs a set of candidate ontologies. At the beginning, this set only contains the inconsistent ontology. Assuming that axiom D was added, for each axiom E sharing symbols with D, a new candidate ontology O_E is created by removing E. Afterwards all candidate ontologies are tested on consistency. If one of the candidate ontologies is consistent, this candidate is a maximal consistent subset of the original ontology. If the all candidate ontologies are inconsistent, new candidate ontologies are created. For each candidate ontology and each axiom structurally connected with D or a previously removed axiom, a new candidate ontology is created by removing the axiom from the candidate ontology. After each step, all candidate ontologies are tested on consistency. This procedure is repeated until a consistent ontology is reached.

4.3.3 Approaches for Dealing with Large ABoxes

The approach for ABox evolution, introduced in the Section 4.2.3, is able to handle large ABoxes. Very large ABoxes are challenging when answering queries as well. When querying knowledge bases with large ABoxes, techniques like ABox summary and ABox partitioning are suggested. Dolby, Fan, Fokoue, Kalyanpur, Kershenbaum, Ma, Murdock, Srinivas, and Welty (2007) introduce an approach to find inconsistencies in large ABoxes. The authors focus on ABoxes with several thousand inconsistencies. As opposed to our approach, where we assume that large parts of the ABox are consistent and the inconsistency is only caused by a very small fragment of the ABox. The technique introduced by Dolby et al. (2007) can be used to determine a consistent subset of the ABox. The basic idea is to determine minimal sets of assertions implying an inconsistency. Those minimal sets are justifications for the inconsistency. As mentioned before, there is a strong connection between justifications and minimal ABox repairs introduced in Section 4.2.1. A minimal ABox repair is a maximal consistent subset of an ABox. Since justifications of inconsistencies are minimal inconsistent sets, removing one assertion from a justification removes the inconsistency caused by this justification from the ABox. In general there can be more than one justification for an inconsistency in an inconsistent ABox. Therefore, to regain consistency, one assertion has to be deleted from each justification. The resulting ABox is called justification-based consistent subset of the original ABox. Note that a justification-based consistent subset does not have to be a maximal consistent subset of the ABox. If all justifications are disjoint, justification-based consistent subsets are maximal consistent subsets. However if not all justifications are disjoint, it is possible that more than one assertion is deleted from one of the justifications leading to non-maximal justification-based consistent subsets. If exactly one assertion is deleted from each justification, the resulting ABox corresponds to a minimal ABox repair. In order to ensure scalability, Dolby et al. (2007) first summarizes the ABox. Intuitively an ABox $\mathcal{A}'$ is a *summarization* of an ABox $\mathcal{A}$ if individuals belonging to the same concepts in $\mathcal{A}$ are integrated into one individual. In general, summarization leads to a much smaller ABox. If such a summarization $\mathcal{A}'$ is consistent w.r.t a TBox $\mathcal{T}$ and an RBox $\mathcal{R}$, the original ABox $\mathcal{A}$ is consistent w.r.t. $\mathcal{T}$ and $\mathcal{R}$ as well (Fokoue, Kershenbaum, Ma, Schonberg, and Srinivas, 2006). However the converse does not hold. In case of a summarized ABox $\mathcal{A}'$ which is inconsistent w.r.t $\mathcal{T}$ and $\mathcal{R}$ it is unclear, if the original ABox is inconsistent w.r.t. $\mathcal{T}$ and $\mathcal{R}$ or if the inconsistency was created during the summary. Therefore, after finding an inconsistency, a justification for this inconsistency is computed. This justification is then scrutinized in order to find out if the inconsistency was caused by the summary. If the inconsistency is a real inconsistency which was already present in the ABox, one assertion is deleted from the justification. In case of an inconsistency caused by the summary process, the respective justification is refined. Refining a justification involves splitting an individual contained in the justification into the respective individuals from the original ABox. After the refinement, consistency is checked again. This refinement step of the summarized ABox followed by consistency checking of the summarized ABox and removal of assertions from a justification of a real inconsistency is repeated until it leads to a consistent ABox. Then the resulting ABox

can be expanded which means to split all individuals which were aggregated during the summary. After this calculation, the resulting ABox is a consistent subset of the original ABox. However it is not guaranteed, that it is a maximal consistent subset. In contrast to this approach, our ABox repair mechanism always creates a maximal consistent subset of the original ABox.

When removing inconsistencies from an ABox, the technique of ABox summary is very helpful. However for ABox deletion, as discussed in this thesis, it is important to keep the names of all individuals. When asking to minimally delete an ABox assertion from a summarized ABox, the minimal deletion would contain information on aggregated individuals.

Example 4.3.6. *Consider the following TBox which is given in form of a DL-clause:*

$$\mathcal{T} = \{D(x) \rightarrow C(x)\}$$

together with the following ABox:

$$\mathcal{A} = \{C(b), C(c), C(e), D(b), D(c), D(e)\}.$$

Since the individuals b, c and e belong to the same concept, they can be summarized into one new individual a_1. Leading to the following summarized ABox:

$$\mathcal{A}' = \{C(a_1), D(a_1)\}.$$

If $C(e)$ is supposed to be deleted from the ABox, the aggregated individual a_1 has to be considered. Deleting $D(a_1)$ leads to deleting $\{C(a_1), D(a_1)\}$. However a_1 is an aggregated individual. Therefore it is unclear which assertions have to be deleted. It would be too much to expand all individuals in $\{C(a_1), D(a_1)\}$ since that would lead to deleting

$$\{C(e), C(b), C(c), D(e), D(b), D(c)\}$$

which corresponds to deleting the whole ABox. However it would be sufficient to only delete $\{C(e), D(e)\}$. Further tests would be necessary to determine which instances of the aggregated individuals are concerned by the deletion.

Even if summarization is very helpful when querying knowledge bases with large ABoxes, as Example 4.3.6 illustrates, summarization complicates the computation of ABox updates. Therefore, summarization is not applied in the approch for ABox evolution presented in the thesis at hand.

4.4 Future Work

One area interesting for future work deals the optimization of the computation of ABox evolution. As mentioned before, we are using a hyper tableau based theorem prover for the computation of $\mathsf{Neg}(\mathcal{A})$-minimal models. A hyper tableau based prover naturally are especially efficient when confronted with horn clauses. However, it is an undeniably fact,

that such a prover is less efficient when confronted with non-horn clauses, since every non-horn clause may cause the prover to branch. Hence, it is desirable to use horn clauses whenever it is possible. Regarding the approach for the computation of ABox evolution introduced in Section 4.2.3, this could be accomplished by influencing the result of the $\mathcal{K}^*$-transformation such that the result is as close to Horn as possible. This can be done by changing the set S given in Definition 4.2.13 according to which the renaming is done. S has to contain at least all concept and role symbols occurring in the ABox. However it is possible to add more concept and role symbols to the set S. Since the set S strongly influences the result of clauses created by (4.2) of Definition 4.2.10 it can be helpful to consider different alternatives for the set S and choose the alternative creating a clause set which is close to Horn clauses.

Another area interesting for future work is the extension of the introduced approach to more expressive description logics. One example would be the extension with qualified number restrictions, leading to the description logic $\mathcal{SHIQ}$. When transforming axioms containing qualified number restrictions into DL-clauses, it is possible that equations are created (Motik et al., 2009). For example the assertion

$$C \sqsubseteq \leq 1R.D$$

leads to the DL-clause

$$C(x) \wedge R(x,y) \wedge R(x,z) \wedge D(y) \wedge D(z) \rightarrow y = z$$

stating that whenever something belonging to the concept C has two R successor belonging to the concept D, then these two R successors are equal. Furthermore, equations are allowed to occur in the ABox as well stating that two individuals are equal. When extending the $\mathcal{K}^*$-transformation to deal with DL-clauses created from qualified number restrictions, it is necessary to handle these equations properly. We believe that this can be done without much changes, but detailed investigations have not been done yet. Furthermore, we do not expect symmetric roles to cause much changes in the $\mathcal{K}^*$-transformation. The fact that a role is symmetric is stated as the following DL-clauses as

$$R(x,y) \rightarrow R(y,x).$$

We expect that this DL-clause can be treated as any other DL-clause during the $\mathcal{K}^*$-transformation. However detailed investigations are still pending.

The extension of the introduced approach to compute changes of the instance level of first-order logic knowledge bases constitutes another interesting direction for future work. Even though the DL-clauses considered in this chapter are very similar to first-order logic clauses, there are some important differences between DL-clauses and first-order logic clauses. One of these differences is the fact that first-order logic clauses are allowed to contain function symbols. Further research is necessary to find out how to handle occurrences of functions within the $\mathcal{K}^*$-transformation.

5 Conclusion

In Section 1, we introduced the vision of the artificial intelligence community, to create machines able to reason on dynamically changing knowledge. To achieve this vision, many obstacles have to be overcome. Of course it is not possible to address all of these problems in this thesis. This is why we focused on two of these obstacles namely

1. the fact that reasoning in logical knowledge bases is intractable and
2. the fact that applying changes to a knowledge base can easily introduce inconsistencies, which cause problems during reasoning.

In this thesis, we focused on knowledge bases given in description logic. For the first challenge we suggested in Chapter 3 the use of knowledge compilation. This technique is well-known in propositional logic and there exist numerous target languages for the precompilation step. In this thesis, one of these target languages, the linkless normal form, is adapted to concepts and TBoxes in the description logic $\mathcal{ALC}$. To accomplish this, several additional normal forms like the $\forall$-normal form, the $\exists$-normal form as well as some variants of the $\exists$-normal form are developed and serve as a basis for the linkless normal form for $\mathcal{ALC}$. Propositional logic formulae in linkless normal form feature some nice properties such as a constant time satisfiability check, a polynomial time test of clausal entailment and a polynomial time projection on a set of propositional logic variables. The linkless normal form for $\mathcal{ALC}$ concepts, was developed such that it exhibits similar properties as its propositional logic counterpart. A linkless concepts C allows

- a constant time satisfiability check,
- a polynomial time check of subsumptions $C \sqsubseteq D$ if D is a q-concept as given in Definition 3.2.30 and
- a polynomial time computation of the uniform interpolant of C with respect to a set of concept symbols.

All these properties are presented in Chapter 3 together with proofs. Furthermore, we showed how to use well-known techniques to construct an approximation of a TBox and how this approximation can be transformed into linkless normal form. The accuracy of the TBox approximation, determines the set of q-concepts for which subsumption checks can be computed efficiently using the linkless version of the TBox approximation. In addition to that, we illustrated how to compute the uniform interpolant of a linkless version of a TBox approximation w.r.t. a set of concept symbols.

The second challenge researched in this thesis is the task of applying changes to the instance level of knowledge bases given in the description logic $\mathcal{SHI}$. In Chapter 4, three different kinds of changes were considered:

- deletion of assertions,
- insertion of assertions and
- repair.

When deleting an assertion from an ABox it is important to ensure that the assertion is not contained in the deductive closure of the resulting knowledge base. When performing an insertion, special care has to be taken that the resulting knowledge base is consistent. In all three cases, the changes performed to the ABox are supposed to be minimal. In order to meet these requirements, three different operations on the instance level of $\mathcal{SHI}$ knowledge bases were introduced. Then the $\mathcal{K}^*$-transformation was developed by adapting an approach called renaming for view deletion in deductive databases (Aravindan and Baumgartner, 2000) to description logic knowledge bases. Based on the result of the $\mathcal{K}^*$-transformation, the different afore-mentioned operations can be computed. To delete an assertion it is necessary to determine $\mathsf{Neg}(\mathcal{A})$-minimal models of the result of the $\mathcal{K}^*$-transformation together with the renamed version of the assertion which has to be deleted. We presented proofs that this $\mathsf{Neg}(\mathcal{A})$-minimal model actually constitutes a minimal deletion as defined afore. Furthermore, we provide proofs that the result of the $\mathcal{K}^*$-transformation can also be used for insertion of assertions and repairing an ABox which is inconsistent w.r.t. its TBox. The $\mathcal{K}^*$-transformation is developed such that it is guided by the ABox of the knowledge base. Since the ABox is likely to be close to a model of the TBox, this can be seen as a semantical guidance. This property of the $\mathcal{K}^*$-transformation ensures that for the computation of a change method only the deviation from the ABox is computed. This is especially beneficial when very large ABoxes are considered and it is assumed that only a small part of the ABox is affected by the changes.

The $\mathcal{K}^*$-transformation was implemented together with methods for deletion of assertions, insertion of assertions and r Chapter 4 presents promising experimental results.

For both challenges we decided to adapt successful techniques used in other logics to description logic knowledge bases. We believe that the transfer of ideas is a fruitful approach since it allows that the different areas of logic can benefit from another. This approach avoids to reinvent the wheel again and again.

Curriculum Vitae

Personal

Name	Claudia Schon, née Obermaier
Email	schon@uni-koblenz.de
Homepage	www.uni-koblenz.de/~obermaie

Education

2000 – 2006 **Diplom**, *Universität Koblenz-Landau*, Koblenz, Studies of computer science with major subject artificial intelligence and theoretical computer science and minor subject mathematics.

1997 – 2000 **Apprenticeship: Information Technology Officer (Informatikkauffrau)**, *Debeka Versicherungsgesellschaft*, Koblenz.

1988 – 1997 **Abitur**, *Kurfürst-Balduin Gymnasium*, Münstermaifeld.

Experience

Vocational

2006 – **Research Assistant**, *Universität Koblenz-Landau, Arbeitsgruppe Künstliche Intelligenz.*
Teaching and research in the RatioLog project.

2001 – 2006 **Student Assistant**, *Universität Koblenz-Landau, Arbeitsgruppe Künstliche Intelligenz.*
Work in different research projects (Trial Solution, In2Math).

Aug. 1997 – Oct. 2000 **Employee at the Debeka Versicherungsgesellschaft**, *Koblenz.*
Apprenticeship: Information Technology Officer (Informatikkauffrau), database programming in COBOL.

Teaching

Supervision of a tutorial group for the lecture "Grundlagen der Theoretischen Informatik", *Universität Koblenz-Landau, Institut für Informatik*, 2012 and 2015.

Supervision of a tutorial group for the lecture "Künstliche Intelligenz 2", *Universität Koblenz-Landau, Institut für Informatik*, 2009 and 2011.

Supervision of a tutorial group for the lecture "Automated Reasoning and Knowledge Representation", *Universität Koblenz-Landau, Institut für Informatik*, 2008 and 2011.

Supervision of a tutorial group for the lecture "Künstliche Intelligenz 1", *Universität Koblenz-Landau, Institut für Informatik*, 2006/07, 2007/08 and 2008/09.

Supervision of a tutorial group for the lecture "KI-Programmierung", *Universität Koblenz-Landau, Institut für Informatik*, 2005/06 and 2006/07.

Supervision of a tutorial group for the lecture "Logik für Informatiker", *Universität Koblenz-Landau, Institut für Informatik*, 2004, 2006 and 2007.

Bibliography

Carlos E. Alchourron, Peter Gärdenfors, and David Makinson. On the logic of theory change: Partial meet contraction and revision functions. *The Journal of Symbolic Logic*, 50:510–530, 1985.

Chandrabose Aravindan and Peter Baumgartner. Theorem proving techniques for view deletion in databases. *Journal of Symbolic Computation*, 29(2):119–147, 2000.

Chandrabose Aravindan and Phan Minh Dung. Knowledge base dynamics, abduction, and database updates. *Journal of Applied Non-Classical Logics*, 5(1):51–76, 1995.

Franz Baader and Werner Nutt. Basic description logics. In Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation, and Applications*, pages 43–95. Cambridge University Press, 2003.

Franz Baader and Rafael Peñaloza. Axiom pinpointing in general tableaux. *Journal of Logic and Computation*, 20(1):5–34, 2010.

Franz Baader and Boontawee Suntisrivaraporn. Debugging SNOMED CT using axiom pinpointing in the description logic $\mathcal{EL}^+$. In *Proceedings of the 3rd Knowledge Representation in Medicine (KR-MED'08): Representing and Sharing Knowledge Using SNOMED*, volume 410 of *CEUR-WS*. CEUR-WS.org, 2008.

Franz Baader, Ralf Küsters, and Ralf Molitor. Computing least common subsumers in description logics with existential restrictions. In Thomas Dean, editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence, IJCAI 99, Stockholm, Sweden*. Morgan Kaufmann, 1999.

Franz Baader, Rafael Peñaloza, and Boontawee Suntisrivaraporn. Pinpointing in the description logic $\mathcal{EL}$. In Joachim Hertzberg, Michael Beetz, and Roman Englert, editors, *KI 2007: Advances in Artificial Intelligence 30th Annual German Conference on AI, KI 2007, Osnabrück, Germany, 2007, Proceedings*, volume 4667 of *Lecture Notes in Artificial Intelligence*. Springer, 2007.

Peter Balsiger and Alain Heuerding. Comparison of theorem provers for modal logics - introduction and summary. In Harrie C. M. de Swart, editor, *Automated Reasoning with Analytic Tableaux and Related Methods International Conference, TABLEAUX'98, Oisterwijk, The Netherlands, 1998, Proceedings*, volume 1397 of *Lecture Notes in Computer Science*. Springer, 1998.

Peter Baumgartner, Ulrich Furbach, and Ilkka Niemelä. Hyper tableaux. In José Júlio Alferes, Luís Moniz Pereira, and Ewa Orlowska, editors, *Logics in Artificial Intelligence, European Workshop, JELIA '96, Évora, Portugal, 1996, Proceedings*, volume 1126 of *Lecture Notes in Computer Science*. Springer, 1996.

Peter Baumgartner, Peter Fröhlich, Ulrich Furbach, and Wolfgang Nejdl. Semantically Guided Theorem Proving for Diagnosis Applications. In Martha E. Pollack, editor, *Proceedings of the 15th International Joint Conference on Artificial Intelligence, IJCAI 97, Nagoya, Japan, 1997, 2 Volumes*, Nagoya, 1997. Morgan Kaufmann.

Peter Baumgartner, Ulrich Furbach, Margret Groß-Hardt, and Alex Sinner. Living book - deduction, slicing, and interaction. *Journal of Automated Reasoning*, 32(3):259–286, 2004.

Peter Baumgartner, Ulrich Furbach, and Björn Pelzer. Hyper tableaux with equality. In Frank Pfennig, editor, *Automated Deduction - CADE 21, 21st International Conference on Automated Deduction, Bremen, Germany, 2007, Proceedings*, volume 4603 of *Lecture Notes in Computer Science*, 2007.

Markus Bender, Björn Pelzer, and Claudia Schon. System description: E-KRHyper 1.4 - extensions for unique names and description logic. In Maria Paola Bonacina, editor, *Automated Deduction - CADE-24, 24th International Conference on Automated Deduction, Lake Placid, NY, USA, 2013, Proceedings*, volume 7898 of *Lecture Notes in Computer Science*. Springer, 2013.

Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, 284(5):34–43, 2001.

Meghyn Bienvenu. Prime implicate normal form for $\mathcal{ALC}$ concepts. In Dieter Fox and Carla P. Gomes, editors, *Proceedings of the 23rd AAAI Conference on Artificial Intelligence, Chicago, Illinois, USA, 2008*. AAAI Press, 2008a.

Meghyn Bienvenu. Prime implicate normal form for $\mathcal{ALC}$ concepts: Long version. Research report IRIT/RR–2008-6–FR, IRIT, Université Paul Sabatier, April 2008b.

Meghyn Bienvenu. On the complexity of consistent query answering in the presence of simple ontologies. In Jörg Hoffmann and Bart Selman, editors, *Proceedings of the 26th AAAI Conference on Artificial Intelligence, 2012, Toronto, Ontario, Canada.* AAAI Press, 2012.

Meghyn Bienvenu and Riccardo Rosati. New inconsistency-tolerant semantics for robust ontology-based data access. In Thomas Eiter, Birte Glimm, Yevgeny Kazakov, and Markus Krötzsch, editors, *Proceedings of the 2013 International Workshop on Description Logics (DL2013), Ulm, Germany, 2013*, volume 1014 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2013a.

Meghyn Bienvenu and Riccardo Rosati. Tractable approximations of consistent query answering for robust ontology-based data access. In Francesca Rossi, editor, *Proceedings of the 23rd International Joint Conference on Artificial Intelligence, IJCAI 2013, Beijing, China, 2013*. IJCAI/AAAI, 2013b.

Meghyn Bienvenu, Camille Bourgaux, and François Goasdoué. Querying inconsistent description logic knowledge bases under preferred repair semantics. In Carla E. Brodley and Peter Stone, editors, *Proceedings of the 28th AAAI Conference on Artificial Intelligence, 2014, Québec City, Québec, Canada.* AAAI Press, 2014.

José A. Blakeley, Per-Åke Larson, and Frank Wm. Tompa. Efficiently updating materialized views. In Carlo Zaniolo, editor, *Proceedings of the 1986 ACM SIGMOD International Conference on Management of Data, Washington, D.C., May 28-30, 1986.*, pages 61–71. ACM Press, 1986. doi: 10.1145/16894.16861. URL `http://doi.acm.org/10.1145/16894.16861`.

Simone Bova, Florent Capelli, Stefan Mengel, and Friedrich Slivovsky. Expander cnfs have exponential DNNF size. *CoRR*, abs/1411.1995, 2014. URL `http://arxiv.org/abs/1411.1995`.

Marco Cadoli and Francesco M. Donini. A survey on knowledge compilation. *AI Communications*, 10(3,4):137–150, 1997.

Diego Calvanese, G. De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Tractable reasoning and efficient query answering in description logics: The dl-lite family. *Journal of Automated Reasoning*, 39:385, 2007.

Diego Calvanese, Evgeny Kharlamov, Werner Nutt, and Dmitriy Zheleznyakov. Updating aboxes in DL-Lite. In Alberto H. F. Laender and Laks V. S. Lakshmanan, editors, *Proceedings of the 4th Alberto Mendelzon International Workshop on Foundations of Data Management, Buenos Aires, Argentina, 2010*, volume 619 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010.

Ashok K. Chandra and George Markowsky. On the number of prime implicants. *Discrete Mathematics*, 24(1):7–11, 1978.

Adnan Darwiche. Compiling devices: A structure-based approach. In Anthony G. Cohn, Lenhart K. Schubert, and Stuart C. Shapiro, editors, *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning, KR 98, Trento, Italy, 1998.* Morgan Kaufmann, 1998a.

Adnan Darwiche. Model-Based Diagnosis using Structured System Descriptions. *Journal of Artificial Intelligence Research*, 8:165–222, 1998b.

Adnan Darwiche. On the tractable counting of theory models and its application to belief revision and truth maintenance. *CoRR*, cs.AI/0003044, 2000.

Adnan Darwiche. Decomposable negation normal form. *Journal of the ACM*, 48(4), 2001.

Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264, 2002.

Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.

Giuseppe De Giacomo, Maurizio Lenzerini, Antonella Poggi, and Riccardo Rosati. On instance-level update and erasure in description logic ontologies. *Journal of Logic and Computation*, 19(5):745–770, 2009.

Maarten De Rijke. Handbook of tableau methods. In Marcello D'Agostino, Dov M. Gabbay, Reiner Hähnle, and Joachim Posegga, editors. *Journal of Logic, Language and Information*, 10(4):518–523, 2001.

Jürgen Dix, Ulrich Furbach, and Ilkka Niemelä. Nonmonotonic reasoning: Towards efficient calculi and implementations. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, pages 1241–1354. Elsevier and MIT Press, 2001.

Julian Dolby, James Fan, Achille Fokoue, Aditya Kalyanpur, Aaron Kershenbaum, Li Ma, William Murdock, Kavitha Srinivas, and Christopher Welty. Scalable cleanup of information extraction data using ontologies. In Karl Aberer, Key-Sun Choi, Natasha Noy, Dean Allemang, Kyung-Il Lee, Lyndon Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Riichiro Mizoguchi, Guus Schreiber, and Philippe Cudré-Mauroux, editors, *The Semantic Web*, volume 4825 of *Lecture Notes in Computer Science*, pages 100–113. Springer, 2007.

William F. Dowling and Jean H. Gallier. Linear-time algorithms for testing the satisfiability of propositional horn formulae. *The Journal of Logic Programming*, 1(3):267–284, 1984.

Thomas Eiter and Georg Gottlob. Propositional circumscription and extended closed world reasoning are π_2^p-complete. *Theoretical Computer Science*, 114:231–245, 1993.

Melvin Fitting. *First-order Logic and Automated Theorem Proving (2Nd Ed.)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1996.

Giorgos Flouris, Dimitris Plexousakis, and Grigoris Antoniou. Generalizing the AGM postulates: preliminary results and applications. In James P. Delgrande and Torsten Schaub, editors, *Proceedings of the 10th International Workshop on Non-Monotonic Reasoning, NMR 2004, Whistler, Canada*, 2004.

Giorgos Flouris, Dimitris Plexousakis, and Grigoris Antoniou. On applying the AGM theory to DLs and OWL. In Yolanda Gil, Enrico Motta, V. Richard Benjamins, and Mark A. Musen, editors, *The Semantic Web - ISWC 2005, 4th International Semantic*

Web Conference, Galway, Ireland, 2005, Proceedings, volume 3729 of *Lecture Notes in Computer Science*. Springer, 2005.

Giorgos Flouris, Zhisheng Huang, Jeff Z. Pan, Dimitris Plexousakis, and Holger Wache. Inconsistencies, negations and changes in ontologies. In Anthony Cohn, editor, *Proceedings, The 21st National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, Boston, Massachusetts, USA, 2006*. AAAI Press, 2006.

Achille Fokoue, Aaron Kershenbaum, Li Ma, Edith Schonberg, and Kavitha Srinivas. The summary abox: Cutting ontologies down to size. In Isabel F. Cruz, Stefan Decker, Dean Allemang, Chris Preist, Daniel Schwabe, Peter Mika, Michael Uschold, and Lora Aroyo, editors, *The Semantic Web - ISWC 2006, 5th International Semantic Web Conference, Athens, GA, USA, 2006*, volume 4273 of *Lecture Notes in Computer Science*. Springer, 2006.

Franz Baader and Diego Calvanese and Deborah L. McGuinness and Daniele Nardi and Peter F. Patel-Schneider, editor. *The Description Logic Handbook: Theory, Implementation, and Applications*, 2003. Cambridge University Press.

Ulrich Furbach and Claudia Obermaier. Applications of automated reasoning. In Christian Freksa, Michael Kohlhase, and Kerstin Schill, editors, *KI 2006: Advances in Artificial Intelligence, 29th Annual German Conference on AI, KI 2006, Bremen, Germany, 2006, Proceedings*, volume 4314 of *Lecture Notes in Computer Science*. Springer, 2007a.

Ulrich Furbach and Claudia Obermaier. Knowledge compilation for description logics. In Joachim Baumeister and Dietmar Seipel, editors, *Proceedings of the 3rd Workshop on Knowledge Engineering and Software Engineering (KESE), Osnabrück, Germany*, volume 282 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007b.

Ulrich Furbach and Claudia Obermaier. Precompiling $\mathcal{ALC}$ tboxes and query answering. In *Proceedings of the 4th Workshop on Contexts and Ontologies, Patras, Greece*, 2008.

Ulrich Furbach and Claudia Schon. Semantically guided evolution of $\mathcal{SHI}$ aboxes. In Didier Galmiche and Dominique Larchey-Wendling, editors, *Automated Reasoning with Analytic Tableaux and Related Methods 22nd International Conference, TABLEAUX 2013, Nancy, France, 2013, Proceedings*, volume 8123 of *Lecture Notes in Computer Science*. Springer, 2013a.

Ulrich Furbach and Claudia Schon. Semantically guided evolution of $\mathcal{SHI}$ aboxes. Reports of the Faculty of Informatics 4/2013, Universität Koblenz-Landau, 2013b. Available at `http://www.uni-koblenz.de/FB4/Publications/Reports`.

Ulrich Furbach and Claudia Schon. Deontic logic for human reasoning. In Thomas Eiter, Hannes Strass, Miroslaw Truszczynski, and Stefan Woltran, editors, *Advances in Knowledge Representation, Logic Programming, and Abstract Argumentation - Essays Dedicated to Gerhard Brewka on the Occasion of His 60th Birthday*, volume 9060 of *Lecture Notes in Computer Science*, pages 63–80. Springer, 2015.

Ulrich Furbach, Heiko Günther, and Claudia Obermaier. A knowledge compilation technique for $\mathcal{ALC}$ tboxes. In H. Chad Lane and Hans W. Guesgen, editors, *Proceedings of the 22nd International Florida Artificial Intelligence Research Society Conference, 2009, Sanibel Island, Florida, USA*. AAAI Press, 2009.

Ulrich Furbach, Ingo Glöckner, and Björn Pelzer. An application of automated reasoning in natural language question answering. *AI Communications*, 23(2-3):241–265, 2010.

Ulrich Furbach, Andrew S. Gordon, and Claudia Schon. Tackling benchmark problems of commonsense reasoning. In Ulrich Furbach and Claudia Schon, editors, *Proceedings of the Workshop on Bridging the Gap between Human and Automated Reasoning - A workshop of the 25th International Conference on Automated Deduction - CADE-25, Berlin, Germany, 2015*, volume 1412 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2015a.

Ulrich Furbach, Claudia Schon, and Frieder Stolzenburg. Cognitive systems and question answering. *Industrie 4.0 Management*, 2015(1):29–32, 2015b.

Ulrich Furbach, Claudia Schon, Frieder Stolzenburg, Karl-Heinz Weis, and Claus-Peter Wirth. The ratiolog project: Rational extensions of logical reasoning. *KI - Künstliche Intelligenz*, 29(3):271–277, 2015c.

Harald Ganzinger, George Hagen, Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. DPLL(T): fast decision procedures. In Rajeev Alur and Doron Peled, editors, *Computer Aided Verification, 16th International Conference, CAV 2004, Boston, MA, USA, 2004, Proceedings*, volume 3114 of *Lecture Notes in Computer Science*. Springer, 2004.

Bernardo Cuenca Grau, Ian Horrocks, Yevgeny Kazakov, and Ulrike Sattler. Just the right amount: Extracting modules from ontologies. In *Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, 2007*, WWW '07. ACM, 2007.

Heiko Günther. Implementation eines Präkompilers für Wissensbasen. Minor Thesis, University of Koblenz-Landau, 2009.

Peter Haase and Ljiljana Stojanovic. Consistent evolution of OWL ontologies. In Asunción Gómez-Pérez and Jérôme Euzenat, editors, *The Semantic Web: Research and Applications, Second European Semantic Web Conference, ESWC 2005, Heraklion, Crete, Greece, 2005, Proceedings*, volume 3532 of *Lecture Notes in Computer Science*. Springer, 2005.

Christian Halaschek-Wiener, Yarden Katz, and Bijan Parsia. Belief base revision for expressive description logics. In Bernardo Cuenca Grau, Pascal Hitzler, Conor Shankey, and Evan Wallace, editors, *Proceedings of the OWLED*06 Workshop on OWL: Experiences and Directions, Athens, Georgia, USA, 2006*, volume 216 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006.

Christian Halaschek-Wiener, Bijan Parsia, and Evren Sirin. Description logic reasoning with syntactic updates. In Robert Meersman and Zahir Tari, editors, *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE, OTM Confederated International Conferences, Montpellier, France, 2006. Proceedings, Part I*, volume 4275 of *Lecture Notes in Computer Science*. Springer, 2006.

Matthew Horridge. *Justification Based Explanation in Ontologies*. PhD thesis, Univesity of Manchester, 2011.

Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Laconic and precise justifications in OWL. In Amit P. Sheth, Steffen Staab, Mike Dean, Massimo Paolucci, Diana Maynard, Timothy W. Finin, and Krishnaprasad Thirunarayan, editors, *The Semantic Web - ISWC 2008, 7th International Semantic Web Conference, ISWC 2008, Karlsruhe, Germany, 2008, Proceedings*, volume 5318 of *Lecture Notes in Computer Science*. Springer, 2008.

Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Explaining inconsistencies in OWL ontologies. In Lluis Godo and Andrea Pugliese, editors, *Scalable Uncertainty Management, Third International Conference, SUM 2009, Washington, DC, USA, 2009. Proceedings*, volume 5785 of *Lecture Notes in Computer Science*. Springer, 2009.

Ian Horrocks. Implementation and optimization techniques. In Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation, and Applications*, pages 306–346. Cambridge University Press, 2003.

Ian Horrocks, Peter F. Patel-Schneider, and Frank Van Harmelen. From $\mathcal{SHIQ}$ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1 (1):2003, 2003.

Peter Jackson and John Pais. Computing prime implicants. In Mark E. Stickel, editor, *10th International Conference on Automated Deduction, Kaiserslautern, FRG, 1990, Proceedings*, volume 449 of *Lecture Notes in Computer Science*. Springer, 1990.

Aditya Kalyanpur. *Debugging and Repair of OWL Ontologies*. PhD thesis, Univesity of Maryland, 2006.

Aditya Kalyanpur, Bijan Parsia, Matthew Horridge, and Evren Sirin. Finding all justifications of OWL DL entailments. In Karl Aberer, Key-Sun Choi, Natasha Fridman Noy, Dean Allemang, Kyung-Il Lee, Lyndon J. B. Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Riichiro Mizoguchi, Guus Schreiber, and Philippe Cudré-Mauroux, editors, *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, 2007*, volume 4825 of *Lecture Notes in Computer Science*. Springer, 2007.

Boris Konev, Dirk Walther, and Frank Wolter. Forgetting and uniform interpolation in large-scale description logic terminologies. In Craig Boutilier, editor, *Proceedings of the*

21st International Joint Conference on Artificial Intelligence, IJCAI 2009, Pasadena, California, USA, 2009. Morgan Kaufmann, 2009.

Marcus Kracht. *Tools and Techniques in Modal Logic*. Elsevier, 1999.

Kevin Lee, Thomas Andreas Meyer, Jeff Z. Pan, and Richard Booth. Computing maximally satisfiable terminologies for the description logic $\mathcal{ALC}$ with cyclic definitions. In Bijan Parsia, Ulrike Sattler, and David Toman, editors, *Proceedings of the 2006 International Workshop on Description Logics (DL2006), Windermere, Lake District, UK, 2006*, volume 189 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006.

Domenico Lembo, Maurizio Lenzerini, Riccardo Rosati, Marco Ruzzi, and Domenico Fabio Savo. Inconsistency-tolerant semantics for description logics. In Pascal Hitzler and Thomas Lukasiewicz, editors, *Web Reasoning and Rule Systems - Fourth International Conference, RR 2010, Bressanone/Brixen, Italy, 2010. Proceedings*, volume 6333 of *Lecture Notes in Computer Science*. Springer, 2010.

Domenico Lembo, Maurizio Lenzerini, Riccardo Rosati, Marco Ruzzi, and Domenico Fabio Savo. Query rewriting for inconsistent dl-lite ontologies. In Sebastian Rudolph and Claudio Gutierrez, editors, *Web Reasoning and Rule Systems - 5th International Conference, RR 2011, Galway, Ireland, 2011. Proceedings*, volume 6902 of *Lecture Notes in Computer Science*. Springer, 2011.

Maurizio Lenzerini and Domenico Fabio Savo. On the evolution of the instance level of dl-lite knowledge bases. In Riccardo Rosati, Sebastian Rudolph, and Michael Zakharyaschev, editors, *Proceedings of the 24th International Workshop on Description Logics (DL 2011), Barcelona, Spain, 2011*, volume 745 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2011.

Fangzhen Lin and Ray Reiter. Forget it! In *In Proceedings of the AAAI Fall Symposium on Relevance*, pages 154–159, 1994.

Hongkai Liu. *Computing Updates in Description Logics*. PhD thesis, Technische Universität Dresden, 2010.

Hongkai Liu, Carsten Lutz, Maja Milicic, and Frank Wolter. Updating description logic aboxes. In Patrick Doherty, John Mylopoulos, and Christopher A. Welty, editors, *Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning, Lake District of the United Kingdom, 2006*. AAAI Press, 2006.

Hongkai Liu, Carsten Lutz, Maja Milicic, and Frank Wolter. Foundations of instance level updates in expressive description logics. *Artificial Intelligence*, 175(18):2170–2197, 2011.

Ying Liu, Wenxiang Gu, Tingting Zou, Qingtao Huang, and Qiao Li. Incremental knowledge compilation using the extension rule. *Journal of Networks*, 8(10), 2013.

John Wylie Lloyd. *Foundations of Logic Programming*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2nd edition, 1993.

Carsten Lutz and Frank Wolter. Foundations for uniform interpolation and forgetting in expressive description logics. In Toby Walsh, editor, *Proceedings of the 22nd International Joint Conference on Artificial Intelligence, IJCAI 2011, Barcelona, Catalonia, Spain, 2011*. IJCAI/AAAI, 2011.

Pierre Marquis. Consequence finding algorithms. In Jürg Kohlas and Serafín Moral, editors, *Handbook of Defeasible Reasoning and Uncertainty Management Systems*, volume 5, pages 41–145. Springer Netherlands, 2000.

Boris Motik. *Reasoning in Description Logics using Resolution and Deductive Databases*. PhD thesis, Univesität Karlsruhe (TH), Karlsruhe, Germany, 2006.

Boris Motik, Rob Shearer, and Ian Horrocks. Optimized reasoning in description logics using hypertableaux. In Frank Pfenning, editor, *Automated Deduction - CADE-21, 21st International Conference on Automated Deduction, Bremen, Germany, 2007, Proceedings*, volume 4603 of *Lecture Notes in Computer Science*. Springer, 2007.

Boris Motik, Rob Shearer, and Ian Horrocks. Hypertableau Reasoning for Description Logics. *Journal of Artificial Intelligence Research*, 36:165–228, 2009.

Neil V. Murray and Erik Rosenthal. Path resolution with link deletion. In Aravind K. Joshi, editor, *Proceedings of the 9th International Joint Conference on Artificial Intelligence, IJCAI 85, Los Angeles, USA, August 1985*. Morgan Kaufmann, 1985.

Neil V. Murray and Erik Rosenthal. Theory links in semantic graphs. In Jörg H. Siekmann, editor, *8th International Conference on Automated Deduction, Oxford, England, 1986, Proceedings*, volume 230 of *Lecture Notes in Computer Science*. Springer, 1986.

Neil V. Murray and Erik Rosenthal. Dissolution: Making paths vanish. *Journal of the ACM*, 40(3):504–535, 1993.

Neil V. Murray and Erik Rosenthal. Tableaux, path dissolution, and decomposable negation normal form for knowledge compilation. In Marta Cialdea Mayer and Fiora Pirri, editors, *Automated Reasoning with Analytic Tableaux and Related Methods, International Conference, TABLEAUX 2003, Rome, Italy, 2003. Proceedings*, volume 2796 of *Lecture Notes in Computer Science*. Springer, 2003.

Ilkka Niemelä. Implementing circumscription using a tableau method. In Wolfgang Wahlster, editor, *12th European Conference on Artificial Intelligence, Budapest, Hungary, 1996, Proceedings*. John Wiley and Sons, Chichester, 1996a.

Ilkka Niemelä. A tableau calculus for minimal model reasoning. In Pierangelo Miglioli, Ugo Moscato, Daniele Mundici, and Mario Ornaghi, editors, *Theorem Proving with Analytic Tableaux and Related Methods, 5th International Workshop, TABLEAUX '96, Terrasini, Palermo, Italy, 1996, Proceedings*, volume 1071 of *Lecture Notes in Computer Science*. Springer, 1996b.

Knot Pipatsrisawat and Adnan Darwiche. Top-down algorithms for constructing structured DNNF: theoretical and practical implications. In Helder Coelho, Rudi Studer, and Michael Wooldridge, editors, *ECAI 2010 - 19th European Conference on Artificial Intelligence, Lisbon, Portugal, 2010, Proceedings*, volume 215 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2010.

Sonja Polster. Implementation of semantically guided evolution of assertional description logic databases. Master's thesis, University of Koblenz-Landau, 2015.

Guilin Qi and Fangkai Yang. A survey of revision approaches in description logics. In Franz Baader, Carsten Lutz, and Boris Motik, editors, *Proceedings of the 2008 International Workshop on Description Logics (DL2008), Dresden Germany, 2008*, volume 353 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

Guilin Qi, Weiru Liu, and David A. Bell. Knowledge base revision in description logics. In Michael Fisher, Wiebe van der Hoek, Boris Konev, and Alexei Lisitsa, editors, *Logics in Artificial Intelligence, 10th European Conference, JELIA 2006, Liverpool, UK, 2006, Proceedings*, volume 4160 of *Lecture Notes in Computer Science*. Springer, 2006.

Alan L. Rector and Ian R. Horrocks. Experience building a large, re-usable medical ontology using a description logic with transitivity and concept inclusions. In *Proceedings of the Workshop on Ontological Engineering, AAAI Spring Symposium (AAAI'97)*. AAAI Press, 1997.

John A. Robinson. Automated deduction with hyper-resolution. *International Journal of Computer Mathematics*, 1(3):227–234, 1965.

Riccardo Rosati. On the complexity of dealing with inconsistency in description logic ontologies. In Toby Walsh, editor, *Proceedings of the 22nd International Joint Conference on Artificial Intelligence, IJCAI 2011, Barcelona, Catalonia, Spain, 2011*. IJCAI/AAAI, 2011.

Erik Rosenthal and Neil V. Murray. Knowledge compilation: Decomposable negation normal form versus linkless formulas. In *Proceedings of the Workshop on Model Computation - Principles, Algorithms, Applications*, 2003.

Sebastian Rudolph, Markus Krötzsch, and Pascal Hitzler. Terminological reasoning in SHIQ with ordered binary decision diagrams. In Dieter Fox and Carla P. Gomes, editors, *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, 2008*. AAAI Press, 2008.

Ulrike Sattler. A concept language extended with different kinds of transitive roles. In Günther Görz and Steffen Hölldobler, editors, *KI-96: Advances in Artificial Intelligence, 20th Annual German Conference on Artificial Intelligence, Dresden, Germany, 1996, Proceedings*, volume 1137 of *Lecture Notes in Computer Science*. Springer, 1996.

Klaus Schild. A correspondence theory for terminological logics: Preliminary report. In John Mylopoulos and Raymond Reiter, editors, *Proceedings of the 12th International Joint Conference on Artificial Intelligence, IJCAI 91, Sydney, Australia, 1991*. Morgan Kaufmann, 1991.

Stefan Schlobach and Ronald Cornet. Non-standard reasoning services for the debugging of description logic terminologies. In Georg Gottlob and Toby Walsh, editors, *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, IJCAI-03, Acapulco, Mexico, 2003*. Morgan Kaufmann, 2003.

Claudia Schon. Linkless normal form for $\mathcal{ALC}$ concepts. Reports of the Faculty of Informatics 12/2010, Universität Koblenz-Landau, 2010. Available at `http://www.uni-koblenz.de/FB4/Publications/Reports`.

Claudia Schon. Linkless normal form for $\mathcal{ALC}$ concepts and tboxes. In Joscha Bach and Stefan Edelkamp, editors, *KI 2011: Advances in Artificial Intelligence, 34th Annual German Conference on AI, Berlin, Germany, 2011. Proceedings*, volume 7006 of *Lecture Notes in Computer Science*. Springer, 2011.

Dana S. Scott. A decision method for validity of sentences in two variables. *Journal of Symbolic Logic*, 27:377, 1962.

Roberto Sebastiani and Michele Vescovi. Automated reasoning in modal and description logics via SAT encoding: the case study of k(m)/$\mathcal{ALC}$-satisfiability. *Journal of Artificial Intelligence Research*, 35:343–389, 2009.

Bart Selman and Henry A. Kautz. Knowledge compilation and theory approximation. *Journal of the ACM*, 43(2):193–224, 1996.

Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical owl-dl reasoner. *Journal of Web Semantics*, 5(2):51–53, 2007.

James R. Slagle, Chin-Liang Chang, and Richard C. T. Lee. A new algorithm for generating prime implicants. *IEEE Transactions on Computers*, 100(4):304–310, 1970.

Robert Stevens, Carole A. Goble, Ian Horrocks, and Sean Bechhofer. Building a bioinformatics ontology using OIL. *IEEE Transactions on Information Technology in Biomedicine*, 6(2):135–141, 2002.

Balder ten Cate, Willem Conradie, Maarten Marx, and Yde Venema. Definitorially complete description logics. In Patrick Doherty, John Mylopoulos, and Christopher A. Welty, editors, *Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning, Lake District of the United Kingdom, 2006*. AAAI Press, 2006.

Stephan Tobies. *Complexity Results and Practical Algorithms for Logics in Knowledge Representation*. PhD thesis, RWTH Aachen University, 2001.

Dmitry Tsarkov and Ian Horrocks. Fact++ description logic reasoner: System description. In Ulrich Furbach and Natarajan Shankar, editors, *Automated Reasoning, Third International Joint Conference, IJCAR 2006, Seattle, WA, USA, 2006, Proceedings*, volume 4130 of *Lecture Notes in Computer Science*. Springer, 2006.

Kewen Wang, Zhe Wang, Rodney Topor, Jeff Z. Pan, and Grigoris Antoniou. Concept and role forgetting in $\mathcal{ALC}$ ontologies. In Abraham Bernstein, David R. Karger, Tom Heath, Lee Feigenbaum, Diana Maynard, Enrico Motta, and Krishnaprasad Thirunarayan, editors, *The Semantic Web - ISWC 2009, 8th International Semantic Web Conference, Washington, DC, USA, 2009, Proceedings*, volume 5318 of *Lecture Notes in Computer Science*. Springer, 2009.

Dmitriy Zheleznyakov, Diego Calvanese, Evgeny Kharlamov, and Werner Nutt. Updating tboxes in DL-lite. In Volker Haarslev, David Toman, and Grant E. Weddell, editors, *Proceedings of the 2010 International Workshop on Description Logics (DL2010), Waterloo, Canada, 2010*, volume 573 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010.

Tingting Zou, Lei Liu, and Shuai Lv. Knowledge compilation for description logic based on concept extension rule. *Journal of Computational Information Systems*, 8(6):2409–2416, 2012.

Index