**Zur Erlangung des akademischen Grades eines Doktors der Wirtschaftswissenschaften (Dr. rer. pol.) von der Fakultät für Wirtschaftswissenschaften der Universität Fridericiana zu Karlsruhe genehmigte Dissertation.**

# Reasoning in Description Logics using Resolution and Deductive Databases

M.Sc. Boris Motik

Tag der mündlichen Prüfung: 09. Januar 2006
Referent:
Prof. Dr. Rudi Studer, Universität Karlsruhe (TH)
Korreferentin 1:
Dr. habil. Ulrike Sattler, University of Manchester
Korreferent 2:
Prof. Dr. Karl-Heinz Waldmann, Universität Karlsruhe (TH)

2006 Karlsruhe

# Abstract

Description logics (DLs) are knowledge representation formalisms with well-understood model-theoretic semantics and computational properties. The DL $\mathcal{SHIQ}(\mathbf{D})$ provides the logical underpinning for the family of Semantic Web ontology languages. Metadata management applications, such as the Semantic Web, often require reasoning with large quantities of assertional information; however, existing algorithms for DL reasoning have been optimized mainly for efficient terminological reasoning. Although techniques for terminological reasoning can be used for assertional reasoning as well, they often do not exhibit good performance in practice.

Deductive databases are knowledge representation systems specifically designed to efficiently reason with large data sets. To see if deductive database techniques can be used to optimize assertional reasoning in DLs, we studied the relationship between the two formalisms. Our main result is a novel algorithm that reduces a $\mathcal{SHIQ}(\mathbf{D})$ knowledge base $KB$ to a disjunctive datalog program $\mathsf{DD}(KB)$ such that $KB$ and $\mathsf{DD}(KB)$ entail the same set of ground facts. In this way, we allow DL reasoning to be performed in $\mathsf{DD}(KB)$ using known, optimized algorithms.

The reduction algorithm is based on several novel results. In particular, we developed a resolution-based algorithm for deciding satisfiability of $\mathcal{SHIQ}$ knowledge bases. Furthermore, to enable representation of concrete data, such as strings or integers, we developed a general approach for reasoning with concrete domains in the framework of resolution, and we use it to extend our algorithms to $\mathcal{SHIQ}(\mathbf{D})$. For unary coding of numbers, these algorithms run in exponential time, and are thus worst-case optimal.

These results allowed us to derive tighter *data complexity* bounds. Namely, assuming that the size of the assertional knowledge dominates the size of the terminological knowledge, the reduction algorithm runs in nondeterministic polynomial time; furthermore, if disjunctions are not used, it runs in deterministic polynomial time.

Finally, we extended these algorithms in several ways. First, we showed that so-called DL-safe rules can be combined with disjunctive programs obtained by the reduction to increase the expressivity of the logic, without affecting decidability. Second, we derived an algorithm for answering conjunctive queries. Third, we extended the algorithms to support metamodeling.

To estimate the practicability of our algorithms, we implemented KAON2—a new DL reasoning system. Our experiments show a performance increase in query answering over existing DL systems of one or more orders of magnitude.

# Contents

# List of Figures

# List of Tables

# Part I

# Foundations

# Chapter 1

# Introduction

Description Logics (DLs) are a family of knowledge representation formalisms that allow representation of domain knowledge and reasoning with it in a formally well-understood way. The first description logic KL-ONE [24] was proposed in order to address the deficiencies of semantic networks [111] and frame-based knowledge representation systems [94]. KL-ONE is based on a model-theoretic semantics, which provides a formal foundation for the vague and imprecise semantics of earlier systems.

Although there are DLs that do not fall into this category, most DLs are fragments of first-order logic. A description logic *terminology* (also called a TBox) describes *concepts* (that is, unary predicates representing sets of individuals) and *roles* (that is, binary predicates representing links between individuals). Concepts can be *atomic*, which means that they are denoted by name, or *complex*, built using constructors that specify necessary and sufficient conditions for concept membership. Apart from a terminology, a description logic *knowledge base* usually has an *assertional* component (also called an ABox), which specifies the membership of individuals or pairs of individuals in concepts and roles, respectively. Historically, the fundamental reasoning problem in description logics is to determine whether one concept *subsumes* another, which is the case if the extension of the former necessarily includes the extension of the latter concept. Apart from subsumption, other reasoning problems, such as checking satisfiability of a knowledge base or retrieving concept individuals, are also important in many applications.

Soon after KL-ONE was introduced, the subsumption problem for KL-ONE concepts was found to be undecidable [130]. From this point on, a large body of description logic research focused on investigating the fundamental trade-offs between expressivity and computational complexity. This line of research culminated in a detailed taxonomy of complexity and undecidability results for various DLs; an overview can be found in [4, Chapter 5].

In parallel, an important goal of description logic research was to develop practical reasoning algorithms and to implement them in practical knowledge representation and reasoning systems. Initially, reasoning algorithms were based on *structural subsumption*. Roughly speaking, such algorithms transform each concept description to

a certain normal form; the structures of normal forms are then compared to decide concept subsumption. After initial experiments with systems based on structural subsumption, such as CLASSIC [22] and LOOM [91], it became evident that sound and complete structural subsumption algorithms are possible only for inexpressive logics.

As a reaction to these deficiencies, tableau algorithms were proposed in [131] as an alternative for description logic reasoning. A tableau algorithm demonstrates satisfiability of a knowledge base by trying to build a model.[1] If such a model can be built, the knowledge base is evidently satisfiable, and if it cannot, the knowledge base is unsatisfiable. Most other reasoning problems can be reduced to satisfiability checking. Tableau algorithms have been built for very expressive logics, so that they are nowadays considered the state of the art for DL reasoning.

$\mathcal{SHIQ}$ [73] is a very expressive description logic, which, apart from the usual Boolean operations on concepts and existential and universal quantification on roles, supports advanced features, such as inverse and transitive roles, role hierarchies, and number restrictions. $\mathcal{SHIQ}(\mathbf{D})$ is an extension of $\mathcal{SHIQ}$ with *datatypes*—a simplified variant of *concrete domains* [5]—, which allow reasoning with concrete data, such as strings or integers. A tableau algorithm for $\mathcal{SHIQ}$ was presented in [73, 74], and it can be easily extended with datatypes in the same way as this was done for the related logic $\mathcal{SHOQ}(\mathbf{D})$ [70]. $\mathcal{SHIQ}(\mathbf{D})$ is important, since it provides the basis of OWL-DL [106]—a W3C recommendation language for ontology representation in the Semantic Web. Namely, OWL-DL is a notational variant of the $\mathcal{SHOIN}(\mathbf{D})$ description logic, which differs from $\mathcal{SHIQ}(\mathbf{D})$ mainly by supporting *nominals*—singleton concepts containing only the specified individual.

Reasoning in $\mathcal{SHIQ}$ is ExpTime-complete [144]. Moreover, tableau algorithms for $\mathcal{SHIQ}$ run in 2NExpTime. Because of their high worst-case complexity, effective optimization techniques are essential to make tableau algorithms usable in practice. Numerous optimization techniques were presented in [66], along with practical evidence of their usefulness. These techniques were implemented in the $\mathcal{SHIQ}$ reasoner FaCT [67], allowing the latter to be successfully applied to practical problems. Another state-of-the-art reasoner for $\mathcal{SHIQ}(\mathbf{D})$ is Racer [59], distinguished from FaCT mainly by supporting assertional knowledge. The reasoner Pellet [105] was implemented recently with the goal of faithfully realizing all the intricacies of the OWL-DL standard.

Description logics were successfully applied to numerous problems, such as information integration [4, Chapter 16] [85, 19, 51], software engineering [4, Chapter 11], and conceptual modeling [4, Chapter 10] [31]. The performance of reasoning algorithms was found to be quite adequate for applications mainly requiring terminological reasoning. However, new applications, such as metadata management in the Semantic Web, require efficient query answering over large ABoxes. So far, attempts have been made to answer queries by a reduction to ABox consistency checking, which can be performed using tableau algorithms. From a theoretical point of view, this approach

[1] For some logics, tableau algorithms actually build finite abstractions of possibly infinite models.

is quite elegant, but from a practical point of view, it has a significant drawback: as the number of ABox individuals increases, the performance becomes quite poor.

We believe that there are two main reasons why tableau algorithms scale poorly to ABox reasoning. First, tableau algorithms treat all individuals separately: to answer a query, a tableau check is needed for each individual to see whether it is an answer to the query. Second, only a small subset of ABox information is usually needed to compute the query answer. These deficiencies have already been acknowledged by the research community, and certain optimization techniques for instance retrieval have been developed [60, 61]. However, the performance of query answering is still not satisfactory in practice.

In parallel to description logic research, many techniques were developed to optimize query answering in deductive databases—a family of knowledge representation formalisms that extend the relational model with deductive features [1, 42]. For example, the first deficiency outlined in the previous paragraph is addressed by managing individuals in sets [1]. This opens the door to various optimization techniques, such as the join-order optimization. Consider the query $worksAt(P, I), hasName(I, \text{`}FZI\text{'})$. It is reasonable to evaluate $hasName(I, \text{`}FZI\text{'})$ first, and then join the result with the tuples in the *worksAt* relation: the second conjunct contains a constant, so evaluating it should return a small number of tuples. Join-order optimizations are usually based on database statistics and are very effective in practice.

The second deficiency can be addressed by identifying the subset of the ABox that is relevant to the query, and then running the reasoning algorithm only on this subset. Magic sets transformation [18] is the primary technique developed to achieve this goal, and it has been used mainly in the context of Horn deductive databases to optimize evaluation of recursive queries. Roughly speaking, the query is modified to ensure that a set of relevant facts is derived during query evaluation; the original query is then evaluated only within this set. The magic sets transformation for disjunctive programs has been presented in [55, 34], along with empirical evidence of its usefulness.

Since techniques for reasoning in deductive databases are now mature, it is natural to investigate whether they can be used to improve query answering over large ABoxes. To facilitate that, we studied the relationship between DLs and disjunctive datalog, with the goal of deriving an algorithm for reducing a $\mathcal{SHIQ}(\mathbf{D})$ knowledge base to a disjunctive datalog program [42] that entails the same set of ground facts as the original knowledge base. Thus, ABox reasoning is reduced to query answering in disjunctive datalog, which allows reusing existing techniques and optimizations for query answering in deductive databases.

The reduction algorithm is based on several novel results, which are interesting in their own right. Next, we overview our contributions:

- In Chapter 5 we present a decision procedure for checking satisfiability of $\mathcal{SHIQ}$ knowledge bases based on basic superposition [14, 96]—a clausal refutation calculus optimized for theorem proving with equality. Parameterized by a suitable term ordering and a selection function, basic superposition decides only a

slightly weaker logic $\mathcal{SHIQ}^-$, in which number restrictions are allowed only on roles not having subroles. For full $\mathcal{SHIQ}$, saturation by basic superposition does not necessarily terminate. To remedy that, we introduce a *decomposition* rule, which transforms certain clauses into simpler ones, thus ensuring termination. We show that decomposition is a very general rule that can be used with any calculus compatible with the standard notion of redundancy [13]. This decision procedure runs in worst-case exponential time, provided that numbers are coded in unary. Unary coding of numbers is standard in description logic algorithms, and, to the best of our knowledge, it is used in all existing reasoning systems. Hence, our algorithms are worst-case optimal under common assumptions.

- Until now, reasoning with concrete domains was predominantly studied in the context of tableau algorithms. Since our algorithms are based on a clausal calculus, existing approaches are not directly applicable to our setting. Therefore, in Chapter 6 we present a general approach for reasoning with a concrete domain in the framework of resolution. Our approach is applicable to any calculus whose completeness proof is based on the model generation method [13], so it can be combined with basic superposition. We apply this approach to the algorithm from Chapter 5 to obtain a procedure for deciding satisfiability of $\mathcal{SHIQ}(\mathbf{D})$ knowledge bases. We show that, assuming a bound on the arity of concrete predicates and an exponential bound on the oracle for concrete domain reasoning, adding datatypes does not increase the complexity of reasoning.

- In Chapter 7 we present an algorithm for reducing a $\mathcal{SHIQ}(\mathbf{D})$ knowledge base to a disjunctive datalog program. Roughly speaking, the algorithms from Chapter 5 and Chapter 6 are first used to compute all nonground consequences of a knowledge base, which are then transformed in a way that allows simulating all remaining ground inferences by basic superposition in disjunctive datalog.

- Based on the algorithm from Chapter 7, in Chapter 8 we analyze the data complexity of reasoning in $\mathcal{SHIQ}(\mathbf{D})$—that is, the complexity measured only in the size of the ABox, while assuming that the TBox is fixed in size. In applications where the size of the ABox is much larger than the size of the TBox, data complexity provides a better estimate of the practical applicability of an algorithm. Surprisingly, the data complexity of satisfiability checking in $\mathcal{SHIQ}(\mathbf{D})$ turns out to be NP-complete, which is better than the ExpTime combined complexity (assuming NP $\subset$ ExpTime). Moreover, we identify the Horn-$\mathcal{SHIQ}(\mathbf{D})$ fragment of $\mathcal{SHIQ}(\mathbf{D})$, which does not provide for modeling disjunctive knowledge, but exhibits polynomial data complexity. This provides theoretical justification for hoping that efficient reasoning with large ABoxes is possible in practice.

- In Chapter 9 we consider a hybrid knowledge representation system consisting of $\mathcal{SHIQ}(\mathbf{D})$ extended with rules. The integration of rules and description logics is achieved by allowing concepts and roles to occur as unary and binary predicates,

respectively, in the atoms of the rule head or body. To achieve decidability, the rules are required to be *DL-safe*: each variable in the rule must occur in a body atom whose predicate is neither a concept nor a role. Intuitively, this makes query answering decidable, since it ensures that rules are applicable only to individuals explicitly occurring in the knowledge base. We show that query answering in such a logic can be performed simply by appending DL-safe rules to the disjunctive datalog program obtained by the reduction.

- In Chapter 10 we extend our algorithms to handle answering and checking subsumption of conjunctive queries [32] over $\mathcal{SHIQ}(\mathbf{D})$ knowledge bases. It is widely believed that conjunctive queries provide a formal foundation for the vast majority of commonly used database queries, so they lend themselves naturally as an expressive query language for description logics.

- In Chapter 11 we consider the problems of extending description logics with metamodeling—a style of modeling that allows concepts to be treated as individuals and vice versa. We show that extending the basic description logic $\mathcal{ALC}$ with metamodeling in the way as this was done in the Semantic Web language OWL-Full [106] leads to undecidability of basic reasoning problems. Therefore, we propose an alternative approach based on HiLog [33]—a logic that aims to simulate second-order reasoning in a first-order framework. We show that, under some minor restrictions, our algorithms can easily be extended to provide a decision procedure for $\mathcal{SHIQ}(\mathbf{D})$ extended with metamodeling.

- To estimate the applicability of our algorithms in practice, we implemented a new DL reasoner KAON2. In Chapter 12 we describe the system architecture, as well as several optimizations required to obtain a system offering competitive performance of reasoning.

- In Chapter 13 we present an evaluation of the performance of KAON2. For answering queries over large ABoxes, our system exhibits performance improvements over Pellet and RACER of one or more orders of magnitude. For TBox reasoning, our system does not match the performance of tableau-based systems; however, it is still capable of solving certain nontrivial problems.

Many of our results were published previously: the resolution decision procedure and the reduction to disjunctive datalog for $\mathcal{SHIQ}^-$ were published in [151]; the decomposition rule and the algorithm for answering conjunctive queries over $\mathcal{SHIQ}$ knowledge bases were published in [152]; the algorithms for reasoning with a concrete domain were published in [150]; reasoning with DL-safe rules was published in [155] and [156]; the results on data complexity were published in [153]; and the results related to metamodeling were published in [154].

# Chapter 2

# Preliminary Definitions

In this chapter we introduce all necessary terminology and recapitulate relevant definitions and results. This chapter is not intended to be of a tutorial nature; please consult the references for a more detailed presentation.

## 2.1 Multi-Sorted First-Order Logic

We recapitulate standard definitions of first-order logic ([46] is a good textbook) extended with multi-sorted signatures.

A *multi-sorted first-order signature* $\Sigma$ is a 4-tuple $(\mathcal{P}, \mathcal{F}, \mathcal{V}, \mathcal{S})$, where $\mathcal{P}$ is a finite set of *predicate symbols*, $\mathcal{F}$ a finite set of *general function symbols*, $\mathcal{V}$ a countable set of *variables*, and $\mathcal{S}$ a finite set of *sorts*. Each predicate and general function symbol is associated with a nonnegative arity $n$. General function symbols of zero arity are called *constants*; all other general function symbols are called simply *function symbols*.[1]

Each $n$-ary general function symbol $f \in \mathcal{F}$ is associated with a sort signature $r_1 \times \ldots \times r_n \rightarrow r$, and each $n$-ary predicate symbol $P \in \mathcal{P}$ is associated with a sort signature $r_1 \times \ldots \times r_n$, for $r_{(i)} \in \mathcal{S}$. The sort of each variable is determined by the function $\mathsf{sort} : \mathcal{V} \rightarrow \mathcal{S}$.

The set of *terms* $\mathcal{T}(\Sigma)$ and the extension of the function $\mathsf{sort}$ to terms are defined as follows: $\mathcal{T}(\Sigma)$ is the smallest set such that $(i)$ $\mathcal{V} \subseteq \mathcal{T}(\Sigma)$, and $(ii)$ if $f \in \mathcal{F}$ has the signature $r_1 \times \ldots \times r_n \rightarrow r$ and $t_i \in \mathcal{T}(\Sigma)$ with $\mathsf{sort}(t_i) = r_i$ for $1 \leq i \leq n$, then $t = f(t_1, \ldots, t_n) \in \mathcal{T}(\Sigma)$ with $\mathsf{sort}(t) = r$. The set of *atoms* $\mathcal{A}(\Sigma)$ is the smallest set such that, if $P \in \mathcal{P}$ has the signature $r_1 \times \ldots \times r_n$ and $t_i \in \mathcal{T}(\Sigma)$ with $\mathsf{sort}(t_i) = r_i$ for $1 \leq i \leq n$, then $P(t_1, \ldots, t_n) \in \mathcal{A}(\Sigma)$. Terms (atoms) not containing variables are called *ground* terms (atoms).

A *position* $p$ is a finite sequence of integers and is usually written as $i_1.i_2 \ldots i_n$. The empty position is denoted with $\epsilon$. If a position $p_1$ is a proper prefix of a position

[1] Many authors do not distinguish constants from function symbols, because this makes the presentation of first-order logic simpler. However, for our results presented in subsequent chapters, this distinction is essential.

$p_2$, then $p_1$ is *above* $p_2$, and $p_2$ is *below* $p_1$. A *subterm* of $t$ at position $p$, written $t|_p$, is defined inductively as $t|_\epsilon = t$ and, if $t = f(t_1, \ldots, t_n)$, then $t|_{i.p} = t_i|_p$. A *replacement* of a subterm of $t$ at position $p$ with the term $s$, written $t[s]_p$, is defined inductively as $t[s]_\epsilon = s$ and, if $t = f(t_1, \ldots, t_n)$, then $t[s]_{i.p} = f(t_1, \ldots, t_i[s]_p, \ldots, t_n)$.

The set of formulae $\mathcal{L}(\Sigma)$ defined over the signature $\Sigma$ is the smallest set such that $\top$ and $\bot$ are in $\mathcal{L}(\Sigma)$, $\mathcal{A}(\Sigma) \subseteq \mathcal{L}(\Sigma)$, and, if $\varphi, \varphi_1, \varphi_2 \in \mathcal{L}(\Sigma)$ and $x \in \mathcal{V}$, then $\neg\varphi$, $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$, $\exists x : \varphi$, and $\forall x : \varphi$ are in $\mathcal{L}(\Sigma)$. As usual, $\varphi_1 \rightarrow \varphi_2$ is an abbreviation for $\neg\varphi_1 \vee \varphi_2$, $\varphi_1 \leftarrow \varphi_2$ is an abbreviation for $\varphi_1 \vee \neg\varphi_2$, and $\varphi_1 \leftrightarrow \varphi_2$ is an abbreviation for $(\varphi_1 \rightarrow \varphi_2) \wedge (\varphi_1 \leftarrow \varphi_2)$. A variable $x$ in a formula $\varphi$ is *free* if it does not occur under the scope of a quantifier. If $\varphi$ does not have free variables, it is *closed*.

The notion of a *subformula* of $\varphi$ at position $p$, written $\varphi|_p$, is defined inductively as $\varphi|_\epsilon = \varphi$; $(\varphi_1 \circ \varphi_2)|_{i.p} = \varphi_i$ for $\circ \in \{\wedge, \vee, \leftarrow, \rightarrow, \leftrightarrow\}$ and $i \in \{1, 2\}$; and $\varphi|_{1.p} = \psi$ for $\varphi = \neg\psi$, $\varphi = \forall x : \psi$, or $\varphi = \exists x : \psi$. A replacement of the subformula $\varphi|_p$ in a formula $\varphi$ with a formula $\psi$ is denoted with $\varphi[\psi]_p$, and is defined in the obvious way.

The *polarity* of the subformula $\varphi|_p$ at position $p$ in a formula $\varphi$, written $\mathsf{pol}(\varphi, p)$, is defined as follows: $\mathsf{pol}(\varphi, \epsilon) = 1$; $\mathsf{pol}(\neg\varphi, 1.p) = -\mathsf{pol}(\varphi, p)$; $\mathsf{pol}(\varphi_1 \circ \varphi_2, i.p) = \mathsf{pol}(\varphi_i, p)$ for $\circ \in \{\wedge, \vee\}$ and $i \in \{1, 2\}$; $\mathsf{pol}(\varphi, 1.p) = \mathsf{pol}(\psi, p)$ for $\varphi = \exists x : \psi$ or $\varphi = \forall x : \psi$; $\mathsf{pol}(\varphi, 1.p) = -\mathsf{pol}(\varphi_1, p)$ and $\mathsf{pol}(\varphi, 2.p) = \mathsf{pol}(\varphi_2, p)$ for $\varphi = \varphi_1 \rightarrow \varphi_2$; finally, $\mathsf{pol}(\varphi_1 \leftrightarrow \varphi_2, i.p) = 0$ for $i \in \{1, 2\}$.

A *substitution* $\sigma$ is a function from $\mathcal{V}$ into $\mathcal{T}(\Sigma)$ such that $\sigma(x) \neq x$ only for a finite number of variables $x$ and, if $\sigma(x) = t$, then $\mathsf{sort}(x) = \mathsf{sort}(t)$. We often write a substitution $\sigma$ as a finite set of mappings $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$. The *empty* substitution (also known as the *identity* substitution), denoted with $\{\}$, is the substitution $\sigma$ such that $x\sigma = x$ for each variable $x$. The result of *applying* a substitution $\sigma$ to a term $t$, written $t\sigma$, is defined recursively as follows: $x\sigma = \sigma(x)$ and $f(t_1, \ldots, t_n)\sigma = f(t_1\sigma, \ldots, t_n\sigma)$. For a substitution $\sigma$ and a variable $x$, the substitution $\sigma_x$ is defined as follows:

$$y\sigma_x = \begin{cases} y\sigma & \text{if } y \neq x \\ y & \text{if } y = x \end{cases}$$

An application of a substitution $\sigma$ to a formula $\varphi$, written $\varphi\sigma$, is defined as follows: $P(t_1, \ldots, t_n)\sigma = P(t_1\sigma, \ldots, t_n\sigma)$; $(\varphi_1 \circ \varphi_2)\sigma = \varphi_1\sigma \circ \varphi_2\sigma$ for $\circ = \{\wedge, \vee, \leftarrow, \rightarrow, \leftrightarrow\}$; $(\neg\varphi)\sigma = \neg(\varphi\sigma)$; $(\forall x : \varphi)\sigma = \forall x : (\varphi\sigma_x)$; and $(\exists x : \varphi)\sigma = \exists x : (\varphi\sigma_x)$.

A *composition* of substitutions $\tau$ and $\sigma$, written $\sigma\tau$, is defined as $x\sigma\tau = (x\sigma)\tau$. A substitution $\sigma$ is called a *variable renaming* if it contains only mappings of the form $x \mapsto y$. A substitution $\sigma$ is equivalent to $\theta$ up to variable renaming if there is a variable renaming $\eta$ such that $\theta = \sigma\eta$; in such a case, $\theta$ is also equivalent to $\sigma$ up to variable renaming [7]. A substitution $\sigma$ is *more general* than a substitution $\theta$ if there is a substitution $\eta$ such that $\theta = \sigma\eta$.

A substitution $\sigma$ is a *unifier* of terms $s$ and $t$ if $s\sigma = t\sigma$. A unifier $\sigma$ of $s$ and $t$ is called a *most general unifier* if it is more general than any other unifier of $s$ and $t$. The notion of unifiers extends to atoms in the obvious way. If a most general unifier $\sigma$ of $s$ and $t$ exists, it is unique up to variable renaming [7], so we write $\sigma = \mathsf{MGU}(s, t)$.

The semantics of multi-sorted first-order logic is defined as follows. An *interpretation* is a pair $I = (\mathcal{D}, \cdot^I)$ where $(i)$ $\mathcal{D}$ is a function assigning to each sort $s \in \mathcal{S}$ an interpretation domain $\mathcal{D}_s$ such that, if $r_i, r_j \in \mathcal{S}$ and $r_i \neq r_j$, then $\mathcal{D}_{r_i} \cap \mathcal{D}_{r_j} = \emptyset$; and $(ii)$ $\cdot^I$ is a function assigning to each predicate symbol $A$ with a signature $r_1 \times \ldots \times r_n$ an interpretation relation $A^I \subseteq \mathcal{D}_{r_1} \times \ldots \times \mathcal{D}_{r_n}$, and to each general function symbol $f$ with a signature $r_1 \times \ldots \times r_n \to r$ an interpretation function $f^I : \mathcal{D}_{r_1} \times \ldots \times \mathcal{D}_{r_n} \to \mathcal{D}_r$. A *variable assignment* is a function $B$ assigning to each variable $x \in \mathcal{V}$ a value from $\mathcal{D}_{\mathsf{sort}(x)}$. An $x$-variant of $B$, denoted with $B_x$, is a variable assignment assigning the same values as $B$ to all variables, except possibly to the variable $x$. The value of a term $t \in \mathcal{T}(\Sigma)$ under $I$ and $B$, written $t^{I,B}$, is defined as follows: if $t = x$, then $t^{I,B} = B(x)$, and, if $t = f(t_1, \ldots, t_n)$, then $t^{I,B} = f^I(t_1^{I,B}, \ldots, t_n^{I,B})$. The truth value of a formula $\varphi$ under $I$ and $B$, written $\varphi^{I,B}$, is defined as follows: $[\top]^{I,B} = \mathit{true}$, $[\bot]^{I,B} = \mathit{false}$, $[P(t_1, \ldots, t_n)]^{I,B} = \mathit{true}$ if and only if $(t_1^{I,B}, \ldots, t_n^{I,B}) \in A^I$; $[\neg\varphi]^{I,B} = \mathit{true}$ if and only if $\varphi^{I,B} = \mathit{false}$; $[\varphi_1 \wedge \varphi_2]^{I,B} = \mathit{true}$ if and only if $\varphi_1^{I,B} = \mathit{true}$ and $\varphi_2^{I,B} = \mathit{true}$; $[\varphi_1 \vee \varphi_2]^{I,B} = \mathit{true}$ if and only if $\varphi_1^{I,B} = \mathit{true}$ or $\varphi_2^{I,B} = \mathit{true}$; $[\exists x : \varphi]^{I,B} = \mathit{true}$ if and only if $\varphi^{I,B_x} = \mathit{true}$ for some $B_x$; and $[\forall x : \varphi]^{I,B} = \mathit{true}$ if and only if $\varphi^{I,B_x} = \mathit{true}$ for all $B_x$. If $\varphi$ is closed, then $\varphi^{I,B}$ does not depend on $B$, so we simply write $\varphi^I$. For a closed formula $\varphi$, an interpretation $I$ is a *model* of $\varphi$, written $I \models \varphi$, if $\varphi^I = \mathit{true}$. A closed formula $\varphi$ is *valid*, written $\models \varphi$, if $I \models \varphi$ for all interpretations $I$; such formulae are also called *tautologies*. Furthermore, $\varphi$ is *satisfiable* if $I \models \varphi$ for at least one interpretation $I$, and $\varphi$ is *unsatisfiable* if no interpretation $I$ exists such that $I \models \varphi$. A closed formula $\varphi_1$ *entails* a formula $\varphi_2$, written $\varphi_1 \models \varphi_2$, if $I \models \varphi_2$ for each interpretation $I$ such that $I \models \varphi_1$. It is well known that $\varphi_1 \models \varphi_2$ if and only if $\varphi_1 \wedge \neg\varphi_2$ is unsatisfiable. Formulae $\varphi_1$ and $\varphi_2$ are *equisatisfiable* if $\varphi_1$ is satisfiable if and only if $\varphi_2$ is satisfiable; $\varphi_1$ and $\varphi_2$ are *equivalent* if the formula $\varphi_1 \leftrightarrow \varphi_2$ is valid.

We often assume that a first-order signature $\Sigma$ contains equality; that is, for each sort $r \in \mathcal{S}$, there is a predicate $\approx_r$ with a sort signature $r \times r$. If the sort is clear from the context, we do not state it explicitly, and simply write $\approx$. An atom $\approx(s, t)$ is usually written as $s \approx t$, and a negated atom $\neg\approx(s, t)$ is usually written as $s \not\approx t$. Models, (un)satisfiability and entailment w.r.t. an *equational theory* are defined as usual, by considering only such models $I$ where all $\approx_r^I$ are equality relations. The latter is the case if $(\alpha, \beta) \in \approx_r^I$ if and only if $\alpha = \beta$, for each $\alpha, \beta \in \mathcal{D}_r$.

Let $\varphi$ be a closed first-order formula and $\Lambda$ a set of positions in $\varphi$. Then $\mathsf{Def}_\Lambda(\varphi)$ is the *definitional normal form* of $\varphi$ with respect to $\Lambda$ and is defined inductively as follows, where $p$ is maximal in $\Lambda \cup \{p\}$ with respect to the prefix ordering on positions, $Q$ is a new predicate not occurring in $\varphi$, the variables $x_1, \ldots, x_n$ are the free variables of $\varphi|_p$, and $\circ$ is $\to$ if $\mathsf{pol}(\varphi, p) = 1$, $\leftarrow$ if $\mathsf{pol}(\varphi, p) = -1$, and $\leftrightarrow$ if $\mathsf{pol}(\varphi, p) = 0$:

$$\begin{aligned} \mathsf{Def}_\emptyset(\varphi) &= \varphi \\ \mathsf{Def}_{\Lambda \cup \{p\}}(\varphi) &= \mathsf{Def}_\Lambda(\varphi[Q(x_1, \ldots, x_n)]_p) \wedge \forall x_1, \ldots, x_n : Q(x_1, \ldots, x_n) \circ \varphi|_p \end{aligned}$$

It is well known [108, 8, 99] that, for any $\Lambda$, the formulae $\varphi$ and $\mathsf{Def}_\Lambda(\varphi)$ are equisatisfiable, and that $\mathsf{Def}_\Lambda(\varphi)$ can be computed in polynomial time.

Let $\varphi$ be a formula and $p$ a position in $\varphi$ such that either $\mathsf{pol}(\varphi, p) = 1$ and $\varphi|_p = \exists x : \psi$, or $\mathsf{pol}(\varphi, p) = -1$ and $\varphi|_p = \forall x : \psi$, where $x, x_1, \ldots, x_n$ are exactly the free variables of $\psi$. Then $\varphi[\psi\{x \mapsto f(x_1, \ldots, x_n)\}]_p$, where $f$ is a new general *Skolem* function symbol not occurring in $\varphi$, is a formula obtained by *skolemization* of $\varphi$ at position $p$. With $\mathsf{sk}(\varphi)$ we denote the formula obtained from $\varphi$ iterative by skolemization at all positions where this is possible. Usually, we assume that $\mathsf{sk}(\varphi)$ is computed by *outer skolemization*, by skolemizing a position $p$ before any position below it. The result of skolemization is unique up to renaming of Skolem function symbols. Formulae $\varphi$ and $\mathsf{sk}(\varphi)$ are equisatisfiable [46].

The subset of ground terms of $\mathcal{F}(\Sigma)$ is called the *Herbrand universe* $HU$ of $\Sigma$. Let $HU_r$ be the subset of $HU$ containing exactly those ground terms $t$ such that $\mathsf{sort}(t) = r$. A *Herbrand interpretation* $I$ is an interpretation such that (*i*) $\mathcal{D}_r = HU_r$; (*ii*) general function symbols are interpreted by themselves—that is, for each $f \in \mathcal{F}$ and $t_i \in HU$, we have $f^I(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$; and (*iii*) $\approx_r^I$ are reflexive, symmetric, transitive, and satisfy the usual equality replacement axioms [46]. The Herbrand base $HB$ of $\Sigma$ is the set of all ground atoms built over the Herbrand universe of $\Sigma$. A Herbrand interpretation can equivalently be considered a subset of the Herbrand base. A formula $\varphi$ is satisfiable if and only if $\mathsf{sk}(\varphi)$ is satisfiable in a Herbrand interpretation [46].

A *multiset* $M$ over a set $N$ is a function $M : N \to \mathbb{N}_0$, where $\mathbb{N}_0$ is the set of all nonnegative integers. A multiset $M$ is finite if $M(x) \neq 0$ for a finite number of $x$; in the remaining sections, we consider only finite multisets. $M$ is empty, written $M = \emptyset$, if $M(x) = 0$ for all $x \in N$. The cardinality of $M$ is defined as $|M| = \Sigma_{x \in N} M(x)$. For two multisets $M_1$ and $M_2$, $M_1 \subseteq M_2$ if $M_1(x) \leq M_2(x)$ for each $x \in N$, and $M_1 = M_2$ if $M_1 \subseteq M_2$ and $M_2 \subseteq M_1$. The union of multisets $M_1$ and $M_2$ is defined as $(M_1 \cup M_2)(x) = M_1(x) + M_2(x)$, the intersection as $(M_1 \cap M_2)(x) = \min(M_1(x), M_2(x))$, and the difference as $(M_1 \setminus M_2)(x) = \max(0, M_1(x) - M_2(x))$.

A *literal* is an atom $A$ or a negated atom $\neg A$. For a literal $L$, we define $\overline{L} = A$ if $L = \neg A$, and $\overline{L} = \neg A$ if $L = A$; that is, $\overline{L}$ is the *complement* of $L$. A *clause* is a multiset of literals and is usually written as $C = L_1 \vee \ldots \vee L_n$. For $n = 1$, $C$ is called a *unit* clause; for $n = 0$, $C$ is the *empty clause*, and is written as $\square$. A clause $C$ is semantically equivalent to $\forall \mathbf{x} : C$, where $\mathbf{x}$ is the set of the free variables of $C$. Satisfiability of clauses is usually considered in a Herbrand interpretation $I$ as follows: for a ground clause $C^G$, $I \models C^G$ if a literal $A_i \in C^G$ exists such that $A_i \in I$, or else a literal $\neg A_j \in C^G$ exists such that $A_j \notin I$; for a nonground clause $C$, $I \models C$ if and only if $I \models C^G$ for each ground instance $C^G$ of $C$. For a first-order formula $\varphi$, $\mathsf{Cls}(\varphi)$ is the set of clauses obtained by *clausifying* $\varphi$—that is, by transforming $\mathsf{sk}(\varphi)$ into conjunctive normal form by exhaustive application of well-known logical equivalences. The formula $\varphi$ is satisfiable if and only if $\mathsf{Cls}(\varphi)$ is satisfiable in a Herbrand model [46]. A variable $x$ in a clause $C$ is *safe* if it occurs in a negative literal of $C$; moreover, $C$ is *safe* if all its variables are safe.

Unless otherwise noted, we denote atoms by letters $A$ and $B$, clauses by $C$ and $D$, literals by $L$, predicates by $P$, $R$, $S$, $T$, and $U$, constants by $a$, $b$, $c$, and $d$, variables by $x$, $y$, and $z$, and terms by $s$, $t$, $u$, $v$, and $w$.

## 2.2 Relations and Orderings

For a set of objects $D$, a binary relation $R$ on $D$ is a subset of $D \times D$. The *inverse* of a relation $R$ is defined as $R^- = \{(y, x) \mid (x, y) \in R\}$. A relation $R$ is $(i)$ *reflexive* if $x \in D$ implies $(x, x) \in R$; $(ii)$ *irreflexive* if $x \in D$ implies $(x, x) \notin R$; $(iii)$ *symmetric* if $R^- \subseteq R$; $(iv)$ *asymmetric* if $(x, y) \in R$ implies $(y, x) \notin R$; $(v)$ *antisymmetric* if $(x, y) \in R$ and $(y, x) \in R$ imply $x = y$; $(vi)$ *transitive* if $(x, y) \in R$ and $(y, z) \in R$ imply $(x, z) \in R$; and $(vii)$ *total* if, for each $x, y \in D$, at least one of $(x, y) \in R$, $(y, x) \in R$, or $x = y$ holds. A $\circ$-closure (where $\circ$ is a combination of the relation properties), written $R^\circ$, is the smallest relation on $D$ such that $R \subseteq R^\circ$ and $\circ$ is satisfied for $R^\circ$. The transitive closure of $R$ is usually written as $R^+$, and the reflexive–transitive closure of $R$ is usually written as $R^*$.

A relation $R$ is *well-founded* if there is no infinite sequence $(\alpha_0, \alpha_1), (\alpha_1, \alpha_2), \ldots$ of pairs in $R$. An object $\alpha \in D$ is in *normal form* w.r.t. $R$ if $R$ does not contain a pair $(\alpha, \beta)$ for any $\beta$; we also say that $\alpha$ is *irreducible w.r.t. R*. *Reducible* is the opposite of irreducible. An object $\beta$ is a normal form of $\alpha$ w.r.t. $R$ if $\beta$ is in normal form w.r.t. $R$ and $(\alpha, \beta) \in R^*$. For a general relation $R$, an object can have none, one, or more normal forms.

A *partial ordering* $\succeq$ over $D$ is a relation on $D$ that is reflexive, antisymmetric, and transitive. A *strict ordering* $\succ$ over $D$ is a relation on $D$ that is irreflexive and transitive. A strict ordering $\succ$ on $D$ can be extended to a strict ordering $\succ_{mul}$ on finite multisets on $D$, called the *multiset extension* of $\succ$, as follows: $M \succ_{mul} N$ if $(i)$ $M \neq N$, and $(ii)$ if $N(x) > M(x)$ for some $x$, then there is some $y \succ x$ such that $M(y) > N(y)$. If $\succ$ is total, then $\succ_{mul}$ is total as well.

For $\succ_i$ orderings on sets $D_i$, $1 \leq i \leq n$, a *lexicographic combination* of $\succ_i$, denoted with $\succ_{lex}$, is an ordering on $D = D_1 \times \ldots \times D_n$ that is defined as follows: $(a_1, \ldots, a_n) \succ_{lex} (b_1, \ldots, b_n)$ if and only if an index $i$ exists, $1 \leq i \leq n$, such that $a_j = b_j$ for $j < i$ and $a_i \succ_i b_i$.

A *term ordering* is an ordering where $D$ is the set of terms $\mathcal{T}(\Sigma)$ for some multi-sorted first-order signature $\Sigma$. A term ordering $\succ$ is *stable under substitutions* if $s \succ t$ implies $s\sigma \succ t\sigma$ for all terms $s$ and $t$, and all substitutions $\sigma$; it is *stable under contexts* if $s \succ t$ implies $u[s]_p \succ u[t]_p$ for all terms $s$, $t$, and $u$, and all positions $p$; it satisfies the *subterm property* if $u[s]_p \succ s$ for all terms $u$ and $s$, and all positions $p \neq \epsilon$. A *rewrite ordering* is an ordering stable under contexts and stable under substitutions; a *reduction ordering* is a well-founded rewrite ordering; and a *simplification ordering* is a reduction ordering with a subterm property.

The *lexicographic path ordering* (LPO) [38, 6] is a term ordering induced by a well-founded strict ordering over general function symbols $>$ (the latter is also called a *precedence*). Each LPO has the subterm property; furthermore, if $>$ is total, the LPO induced by $>$ is total on ground terms. It is defined as follows:

$s \succ_{lpo} t$ if

1. $t$ is a variable occurring as a proper subterm of $s$, or
2. $s = f(s_1, \ldots, s_m)$, $t = g(t_1, \ldots, t_n)$, and at least one of the following holds:
   (a) $f > g$ and, for all $i$ with $1 \leq i \leq n$, we have $s \succ_{lpo} t_i$, or
   (b) $f = g$ and, for some $j$, we have $(s_1, \ldots, s_{j-1}) = (t_1, \ldots t_{j-1})$, $s_j \succ_{lpo} t_j$, and $s \succ_{lpo} t_k$ for all $k$ with $j < k \leq n$, or
   (c) $s_j \succeq_{lpo} t$ for some $j$ with $1 \leq j \leq m$.

## 2.3 Rewrite Systems

An excellent textbook introduction to rewrite systems can be found in [6], and an overview of the major results can be found in [38]. A *rewrite system* $R$ is a set of rewrite rules $s \Rightarrow t$ where $s$ and $t$ are terms. A *rewrite relation* induced by $R$, denoted with $\Rightarrow_R$, is the smallest relation such that $s \Rightarrow t \in R$ implies $u[s\sigma]_p \Rightarrow_R u[t\sigma]_p$ for all terms $s$, $t$, and $u$, all substitutions $\sigma$, and all positions $p$. For two terms $s$ and $t$, we write $s \Downarrow_R t$ if there is a term $u$ such that $s \Rightarrow_R^* u$ and $t \Rightarrow_R^* u$, where $\Rightarrow_R^*$ is the reflexive–transitive closure of $\Rightarrow_R$. A rewrite system $R$ is *confluent* if $\Downarrow_R$ and $\Leftrightarrow_R^*$ coincide, where $\Leftrightarrow_R^*$ is the symmetric–reflexive–transitive closure of $\Rightarrow_R$. For a confluent, well-founded rewrite system $R$, each element $\alpha$ has a unique normal form w.r.t. $\Rightarrow_R$, which we denote with $\mathsf{nf}_R(\alpha)$.

For a confluent well-founded rewrite system $R$ consisting of ground rewrite rules only, $R^*$ is the smallest set of ground equalities $s \approx t$ such that, for all ground terms $s$ and $t$, if $\mathsf{nf}_R(s) = \mathsf{nf}_R(t)$, then $s \approx t \in R^*$.

## 2.4 Ordered Resolution

*Ordered resolution* [13] is one of the most widely used calculi for theorem proving in first-order logic. The rules of the calculus are parameterized with an admissible ordering $\succ$ on literals and a selection function.

An ordering on literals $\succ$ is *admissible* if (*i*) it is well-founded, stable under substitutions, and total on ground literals; (*ii*) $\neg A \succ A$ for all ground atoms $A$; and (*iii*) $B \succ A$ implies $B \succ \neg A$ for all atoms $A$ and $B$. A literal $L$ is (strictly) maximal with respect to a clause $C$ if there is no literal $L' \in C$ such that $L' \succ L$ ($L' \succeq L$). A literal $L \in C$ is (strictly) maximal in $C$ if and only if $L$ is (strictly) maximal with respect to $C \setminus L$. By taking its multiset extension, each ordering on literals $\succ$ can be extended to an ordering on clauses, which we ambiguously denote with $\succ$ as well. Because the literal ordering is total and well-founded on ground literals, the clause ordering is total and well-founded on ground clauses.

A *selection function* $S$ assigns to each clause $C$ a possibly empty subset of negative literals of $C$; the literals in $S(C)$ are said to be *selected*. No other restrictions are imposed on the selection function.

With $\mathcal{R}$ we denote the ordered resolution calculus, consisting of the following inference rules, where the clauses $C \vee A \vee B$ and $D \vee \neg B$ are called the *main premises*, $C \vee A$ is called the *side premise*, and $C\sigma \vee A\sigma$ and $C\sigma \vee D\sigma$ are called *conclusions* (as usual in resolution theorem proving, we make a technical assumption that the premises do not have variables in common):

**Positive factoring:**
$$\frac{C \vee A \vee B}{C\sigma \vee A\sigma}$$

where $(i)$ $\sigma = \mathsf{MGU}(A, B)$, $(ii)$ $A\sigma$ is strictly maximal with respect to $C\sigma \vee B\sigma$, and no literal is selected in $C\sigma \vee A\sigma \vee B\sigma$.

**Ordered resolution:**
$$\frac{C \vee A \quad D \vee \neg B}{C\sigma \vee D\sigma}$$

where $(i)$ $\sigma = \mathsf{MGU}(A, B)$, $(ii)$ $A\sigma$ is strictly maximal with respect to $C\sigma$, and no literal is selected in $C\sigma \vee A\sigma$, $(iii)$ $\neg B\sigma$ is either selected in $D\sigma \vee \neg B\sigma$, or it is maximal with respect to $D\sigma$ and no literal is selected in $D\sigma \vee \neg B\sigma$.

It is important to distinguish an *inference rule* from an *inference*. An inference rule can be understood as a template that specifies actions to be applied to any premises. An inference is an application of an inference rule to concrete premises. An inference $\xi'$ is an *instance* of an inference $\xi$ if $\xi'$ is obtained by applying a substitution $\sigma$ to all premises and the conclusion of $\xi$; the inference $\xi'$ is also written as $\xi\sigma$. An inference is *ground* if all its clauses are ground.

Ordered resolution is compatible with powerful *redundancy elimination techniques*, which allow deleting certain clauses during the theorem proving process without loss of completeness. A ground clause $C$ is *redundant* in a set of ground clauses $N$ if there are clauses $D_i \in N$, $1 \leq i \leq n$, such that $C \succ D_i$ for all $i$, and $D_1, \ldots, D_n \models C$. A ground inference $\xi$ of $\mathcal{R}$ with premises $C_1$ and $C_2$, and a conclusion $C$ is *redundant* in a set of ground clauses $N$ if there are clauses $D_i \in N$, $1 \leq i \leq n$, such that $\max(C_1, C_2) \succ D_i$ and $C_1, C_2, D_1, \ldots, D_n \models C$, where $\max(C_1, C_2)$ is the larger clause of $C_1$ and $C_2$ w.r.t. $\succ$. A nonground clause $C$ (inference $\xi$) is redundant in a nonground set of clauses $N$ if each ground instance of $C$ ($\xi$) is redundant in the set of ground instances of $N$. A set of clauses $N$ is *saturated* by $\mathcal{R}$ up to redundancy if each inference by $\mathcal{R}$ from premises in $N$ is redundant in $N$. Ordered resolution is sound and complete: if a set of clauses $N$ is saturated up to redundancy by $\mathcal{R}$, then $N$ is satisfiable if and only if it does not contain the empty clause.

If a clause $C$ is a tautology, then $C$ is redundant in any set of clauses $N$. A sound and complete tautology check would itself require theorem proving, and would therefore be difficult to realize. Therefore, in practice one usually only checks for *syntactic tautologies*, which are clauses containing a pair of literals $A$ and $\neg A$.

A clause $C$ *subsumes* a clause $D$ if there is a substitution $\sigma$ such that $C\sigma \subseteq D$ and $|C| < |D|$. If a clause $C$ is subsumed by a clause from a set of clauses $N$, then $C$ is redundant in $N$.

A *theorem proving derivation* by $\mathcal{R}$ from a set of clauses $N$ is a sequence of sets of clauses $N = N_0, N_1, \ldots$ such that, for each $i > 0$, either $(i)$ $N_{i+1} = N_i \cup \{C\}$ where $C$ is the conclusion of an inference by $\mathcal{R}$ from premises in $N_i$, or $(i)$ $N_{i+1} = N_i \setminus \{C\}$ where $C$ is redundant in $N_i$. A *refutation for* $N$ is a derivation from $N$ such that some $N_j$ contains the empty clause. A derivation is *fair* with *limit* $N_\infty = \bigcup_j \bigcap_{k \geq j} N_k$ if each clause $C$ that can be deduced from nonredundant premises in $N_\infty$ is contained in some set $N_j$. In [13] it was shown that under the standard notion of redundancy, each inference from premises in $N_\infty$ is redundant in $N_\infty$.

Hence, unsatisfiability of a set of clauses $N$ can be demonstrated by a fair derivation from $N$. If $N$ is unsatisfiable, then we shall eventually derive the empty clause; if $N$ is satisfiable, then the limit of the derivation $N_\infty$ does not contain the empty clause.

## 2.5 Basic Superposition

In order to deal with first-order theories containing equality, ordered resolution was extended in [120] to *paramodulation*—a calculus with explicit rules for equality reasoning. A refinement of paramodulation, known as *superposition*, was presented in [9], where ordering restrictions restrict certain unnecessary inferences. Further optimizations of paramodulation and superposition were presented in [14]. These optimizations are very general, but a simplified version of the calculus, called *basic superposition*, was presented in [10, 12]. A very related calculus, based on an inference model with constrained clauses, was presented in [96].

The idea of basic superposition is to render superposition inferences into terms introduced by previous unification steps redundant. In practice, this technique has been shown essential for solving some particularly difficult problems in first-order logic with equality [92]. Furthermore, basic superposition shows that superposition into arguments of Skolem function symbols is not necessary for completeness. Namely, any Skolem function symbol $f$ occurs in the initial clause set with variable arguments, so, in any term $f(t)$, if $t$ is not a variable, it was introduced by a previous unification step.

It is common practice in equational theorem proving to consider logical theories containing only the equality predicate. This simplifies the theoretical treatment without loss of generality. Literals $P(t_1, \ldots, t_n)$, where $P$ is not the equality predicate, are encoded as $P(t_1, \ldots, t_n) \approx \mathsf{T}$, where $\mathsf{T}$ is a new propositional symbol. Thus, predicate symbols actually become general function symbols. It is well known that this transformation preserves satisfiability. To avoid considering terms where predicate symbols occur as proper subterms, one usually employs a multi-sorted framework, where all predicate symbols and the symbol $\mathsf{T}$ are of one sort, which is different from the sort of general function symbols and variables. We consider $P(t_1, \ldots, t_n)$ to be a syntactic shortcut for $P(t_1, \ldots, t_n) \approx \mathsf{T}$. To avoid ambiguity, we use the following terminology: first-order terms (general function symbols) obtained by the encoding are called

*$\mathcal{E}$-terms* (*$\mathcal{E}$-general function symbols*), *predicate symbols* are $\approx$ and the $\mathcal{E}$-general function symbols corresponding to predicate symbols before encoding, whereas *constants* (*function symbols*) are $\mathcal{E}$-general function symbols corresponding to constants (function symbols) before encoding. For example, the literal $P(c, f(x))$ is a shortcut for $P(c, f(x)) \approx \mathsf{T}$; furthermore, $P(c, f(x))$ is an $\mathcal{E}$-term containing $\mathcal{E}$-general function symbols $P$, $c$, and $f$; however, $P$ is a predicate symbol, $f$ is a function symbol, $c$ is a constant, and only $c$, $f(x)$, and $x$ are terms.

Furthermore, it is common to assume that the predicate $\approx$ has built-in symmetry: a literal $s \approx t$ should also be interpreted as $t \approx s$ (the same holds for negative equality literals as well).

The inference rules of basic superposition are formulated by breaking a clause into two parts: $(i)$ the *skeleton* clause $C$ and $(ii)$ the substitution $\sigma$ representing the cumulative effects of previous unifications. These two components together are called a *closure*, which is written as $C \cdot \sigma$ and is logically equivalent to a clause $C\sigma$. A closure $C \cdot \sigma$ can, for convenience, equivalently be represented as $C\sigma$, where the terms occurring at variable positions of $C$ are *marked*[2] by $[\,]$. Any position at or below a marked position is called a *substitution position*. Note that all variables of $C\sigma$ occur at substitution positions, so we do not mark them for readability purposes.

The following closure is logically equivalent to the clause $P(f(y)) \vee g(b) \approx b$. On the left-hand side, the closure is represented by a skeleton and a substitution explicitly, whereas, on the right-hand side, it is represented by marking the positions of variables in the skeleton.

$$(P(x) \vee z \approx b) \cdot \{x \mapsto f(y), z \mapsto g(b)\} \equiv P([f(y)]) \vee [g(b)] \approx b \tag{2.1}$$

A closure $C \cdot \sigma$ is *ground* if $C\sigma$ is ground. To technically simplify the presentation, we consider each closure to be in the *standard form*, which is the case if $(i)$ the substitution $\sigma$ does not contain trivial mappings of the form $x \mapsto y$, and $(ii)$ all variables from $\mathsf{dom}(\sigma)$ occur in $C$. A closure $C \cdot \sigma$ can be brought into the standard form in the following way: if $x \mapsto t$ is a mapping in $\sigma$ that violates the conditions of the standard form, then let $\sigma'$ be $\sigma \setminus \{x \mapsto t\}$, and replace $C \cdot \sigma$ with $C\{x \mapsto t\} \cdot \sigma'\{x \mapsto t\}$.

A closure $(C\sigma_1) \cdot \sigma_2$ is a *retraction* of a closure $C \cdot \sigma$ if $\sigma = \sigma_1\sigma_2$. Intuitively, a retraction is obtained by moving some marked positions lower in the closure. For example, the following is a retraction of the closure (2.1):

$$(P(x) \vee g(z) \approx b) \cdot \{x \mapsto f(y), z \mapsto b\} \equiv P([f(y)]) \vee g([b]) \approx b \tag{2.2}$$

**Parameters for Basic Superposition.** Basic superposition is parameterized with a selection function $S$, which is defined exactly as for ordered resolution. However, whereas ordered resolution is parameterized with an ordering on literals, basic superposition is parameterized with an ordering $\succ$ on $\mathcal{E}$-terms. Such an ordering is

[2] In [14], terms at marked positions are enclosed in a frame. We decided to use a different notation, because framing introduced problems with text layout. Our notation should not be confused with the notation for modalities in multi-modal logic.

*admissible* for basic superposition if it is a reduction ordering total on ground terms and $\mathsf{T}$ is the smallest element. An ordering $\succ$ can be extended to an ordering on literals (ambiguously denoted with $\succ$ as well) by identifying each positive literal $s \approx t$ with a multiset $\{\{s\},\{t\}\}$ and each negative literal $s \not\approx t$ with a multiset $\{\{s,t\}\}$, and by comparing these multisets using a two-fold multiset extension $(\succ_{mul})_{mul}$ of $\succ$. The literal ordering obtained in such a way is total on ground literals. The literal $L \cdot \sigma$ is *(strictly) maximal* with respect to a closure $C \cdot \sigma$ if there is no literal $L' \in C$ such that $L'\sigma \succ L\sigma$ ($L'\sigma \succeq L\sigma$) (observe that this definition does not assume that $L \in C$). Similarly, for a closure $C \cdot \sigma$ and a literal $L \in C$, the literal $L \cdot \sigma$ is (strictly) maximal in $C \cdot \sigma$ if and only if it is (strictly) maximal with respect to $(C \setminus L) \cdot \sigma$.

**Inference Rules.** In the rules of basic superposition, we make the technical assumption that all premises are variable disjoint, and that they are expressed using the same substitution. A literal $L \cdot \theta$ is *(strictly) eligible for superposition* in a closure $(C \vee L) \cdot \theta$ if there are no selected literals in $(C \vee L) \cdot \theta$ and $L \cdot \theta$ is (strictly) maximal with respect to $C \cdot \theta$. A literal $L \cdot \theta$ is *eligible for resolution* in a closure $(C \vee L) \cdot \theta$ if it is selected in $(C \vee L) \cdot \theta$, or there are no selected literals in $(C \vee L) \cdot \theta$ and $L \cdot \theta$ is maximal with respect to $C \cdot \theta$. The basic superposition calculus, $\mathcal{BS}$ for short, consists of the following rules:

**Positive superposition:**
$$\frac{(C \vee s \approx t) \cdot \rho \quad (D \vee w \approx v) \cdot \rho}{(C \vee D \vee w[t]_p \approx v) \cdot \theta}$$

where $(i)$ $\sigma = \mathsf{MGU}(s\rho, w\rho|_p)$ and $\theta = \rho\sigma$, $(ii)$ $t\theta \not\succeq s\theta$ and $v\theta \not\succeq w\theta$, $(iii)$ $(s \approx t) \cdot \theta$ is strictly eligible for superposition in $(C \vee s \approx t) \cdot \theta$, $(iv)$ $(w \approx v) \cdot \theta$ is strictly eligible for superposition in $(D \vee w \approx v) \cdot \theta$, $(v)$ $s\theta \approx t\theta \not\succeq w\theta \approx v\theta$, $(vi)$ $w|_p$ is not a variable.

**Negative superposition:**
$$\frac{(C \vee s \approx t) \cdot \rho \quad (D \vee w \not\approx v) \cdot \rho}{(C \vee D \vee w[t]_p \not\approx v) \cdot \theta}$$

where $(i)$ $\sigma = \mathsf{MGU}(s\rho, w\rho|_p)$ and $\theta = \rho\sigma$, $(ii)$ $t\theta \not\succeq s\theta$ and $v\theta \not\succeq w\theta$, $(iii)$ $(s \approx t) \cdot \theta$ is strictly eligible for superposition in $(C \vee s \approx t) \cdot \theta$, $(iv)$ $(w \not\approx v) \cdot \theta$ is eligible for resolution in $(D \vee w \not\approx v) \cdot \theta$, $(v)$ $w|_p$ is not a variable.

**Reflexivity resolution:**
$$\frac{(C \vee s \not\approx t) \cdot \rho}{C \cdot \theta}$$

where $(i)$ $\sigma = \mathsf{MGU}(s\rho, t\rho)$ and $\theta = \rho\sigma$, $(ii)$ $(s \not\approx t) \cdot \theta$ is eligible for resolution in $(C \vee s \not\approx t) \cdot \theta$.

**Equality factoring:**
$$\frac{(C \vee s \approx t \vee s' \approx t') \cdot \rho}{(C \vee t \not\approx t' \vee s' \approx t') \cdot \theta}$$

where $(i)$ $\sigma = \mathsf{MGU}(s\rho, s'\rho)$ and $\theta = \rho\sigma$, $(ii)$ $t\theta \not\succeq s\theta$ and $t'\theta \not\succeq s'\theta$, $(iii)$ $(s \approx t) \cdot \theta$ is eligible for superposition in $(C \vee s \approx t \vee s' \approx t') \cdot \theta$.

**Ordered Hyperresolution:** $$\frac{E_1 \dots E_n \quad N}{(C_1 \vee \dots \vee C_n \vee D) \cdot \theta}$$

where $(i)$ $E_i$ are of the form $(C_i \vee A_i) \cdot \rho$, for $1 \leq i \leq n$, $(ii)$ $N$ is of the form $(D \vee \neg B_1 \vee \dots \vee \neg B_n) \cdot \rho$, $(iii)$ $\sigma$ is the most general substitution such that $A_i\theta = B_i\theta$ for $1 \leq i \leq n$ and $\theta = \rho\sigma$, $(iv)$ each $A_i \cdot \theta$ is strictly eligible for superposition in $E_i$, $(v)$ either $\neg B_i \cdot \theta$ are selected, or nothing is selected, $n = 1$, and $\neg B_1 \cdot \theta$ is maximal w.r.t. $D \cdot \theta$.

In an inference by ordered hyperresolution, the closures $E_i$ are called the *electrons* or the *side premises*, and the closure $N$ is called the *nucleus* or the *main premise*. $\mathcal{BS}$ was presented in [14, 96] without the hyperresolution rule. However, as noted in [9], hyperresolution is analogous to a macro: it combines the effects of $n$ negative superpositions of $(A_i \approx \mathsf{T}) \cdot \rho$ from $E_i$ into $(B_i \not\approx \mathsf{T}) \cdot \rho$ of $N$, resulting in $(\mathsf{T} \not\approx \mathsf{T}) \cdot \theta$, which is immediately eliminated by reflexivity resolution. Furthermore, note that a positive superposition of a main premise into a positive literal $(B \approx \mathsf{T}) \cdot \rho$ results in a tautology $(\mathsf{T} \approx \mathsf{T}) \cdot \theta$, which can be deleted. Hence, ordered hyperresolution captures all inferences involving several premises and literals with predicates other than $\approx$. One might also consider ordered factoring, which combines equality resolution on $(C \vee A \approx \mathsf{T} \vee B \approx \mathsf{T}) \cdot \rho$ with reflexivity resolution. We decided not to do this to keep the presentation simpler.

Basic superposition is a sound and complete refutation calculus: for $N$ a set of closures saturated up to redundancy, $N$ is unsatisfiable if and only if it contains the empty closure.

**Completeness of Basic Superposition.** We now briefly overview the completeness proof of basic superposition. We base our presentation on the proof by Nieuwenhuis and Rubio from [96, 97], which is compatible with the one from [14].

The literal ordering $\succ$ is extended to closures by a multiset extension, where closures are treated as multisets of literals. We denote such an ordering on closures by $\succ$ as well. Because the literal ordering is total on ground literals, the closure ordering is total on ground closures.

Let $C \cdot \sigma$ be a closure and $\tau$ a ground substitution. The set of *succedent-top-left variables* of $C \cdot \sigma$ w.r.t. $\tau$, written $\mathsf{stlvars}(C \cdot \sigma, \tau)$, is the set of all variables $x$ occurring in a literal $x \approx s \in C$ such that $x\sigma\tau \succ s\sigma\tau$.

Let $R$ be a ground and convergent rewrite system and $\tau$ a ground substitution. A variable $x$ occurring in the skeleton $C$ of a closure $C \cdot \sigma$ is *variable irreducible w.r.t. $R$* if $(i)$ $x\sigma\tau$ is irreducible by $R$, or $(ii)$ $x \in \mathsf{stlvars}(C \cdot \sigma, \tau)$ and, for all $x \approx s \in C$, $x\sigma\tau$ is irreducible by those rules $l \Rightarrow r$ from $R$ for which $x\sigma\tau \approx s\sigma\tau \succ l \approx r$. A ground instance $C \cdot \sigma\tau$ is *variable irreducible w.r.t. $R$* if all variables $x$ from $C$ are variable irreducible w.r.t. $R$. Let $\mathsf{irred}_R(C \cdot \sigma)$ be the set of all variable irreducible ground instances of $C \cdot \sigma$ w.r.t. $R$. For a set of closures $N$, let $\mathsf{irred}_R(N)$ be the set of all variable irreducible ground instances of closures in $N$ w.r.t. $R$. Finally, let

$\mathsf{irred}_R(N)^{\prec D}$ be the subset of closures of $\mathsf{irred}_R(N)$ smaller than a ground closure $D$ (w.r.t. the ordering $\prec$ on closures).

Let $\xi$ be a $\mathcal{BS}$ inference with premises $D_1 \cdot \sigma$ and $D_2 \cdot \sigma$, and a conclusion $C \cdot \rho$; $R$ a rewrite system; and $\tau$ a ground substitution such that $\xi\tau$ is a ground instance of $\xi$. Then, $\xi\tau$ is *variable irreducible w.r.t. $R$* if all $D_1 \cdot \sigma\tau$, $D_2 \cdot \sigma\tau$, and $C \cdot \rho\tau$ are variable irreducible w.r.t. $R$.

The notion of redundancy for $\mathcal{BS}$ is defined as follows. A closure $C \cdot \sigma$ is *redundant in* $N$ if, for all rewrite systems $R$ and all ground substitutions $\tau$ such that $C \cdot \sigma\tau$ is variable irreducible w.r.t. $R$, we have $R \cup \mathsf{irred}_R(N)^{\prec C \cdot \sigma\tau} \models C \cdot \sigma\tau$. An inference $\xi$ with premises $D_1 \cdot \sigma$ and $D_2 \cdot \sigma$, and a conclusion $C \cdot \rho$ is redundant in $N$ if, for all rewrite systems $R$ and all ground substitutions $\tau$ such that $\xi\tau$ is a variable irreducible ground instance of $\xi$ w.r.t. $R$, we have $R \cup \mathsf{irred}_R(N)^{\prec D} \models C \cdot \rho\tau$, for $D = \max(D_1 \cdot \sigma\tau, D_2 \cdot \sigma\tau)$. The set of closures $N$ is saturated up to redundancy by $\mathcal{BS}$ if all inferences from premises in $N$ are redundant in $N$.

A set of closures $N$ is *well-constrained* if $\mathsf{irred}_R(N) \cup R \models N$ for any rewrite system $R$. If, for all $C \cdot \rho \in N$, $\rho$ is the empty substitution, then $N$ is well-constrained: any variable reducible position of a ground instance of $C \cdot \rho$ can be reduced with rules from $R$ to a closure in $\mathsf{irred}_R(N)$. Furthermore, if $N'$ is obtained from a well-constrained set $N$ by a sound inference rule, then $N'$ is also well-constrained.

Let $N$ be the set of closures obtained by saturating a well-constrained set $N_0$ up to redundancy by $\mathcal{BS}$. Then, $N$ is satisfiable if it does not contain the empty closure. Namely, using a variant of the model building technique [14, 96], one can generate a ground convergent rewrite system $R_N$, which uniquely defines the Herbrand interpretation ${R_N}^*$ such that ${R_N}^* \models \mathsf{irred}_{R_N}(N)$. Finally, since $N_0$ is well-constrained, $N$ is well-constrained as well. Since $R_N \subseteq {R_N}^*$, it follows that ${R_N}^* \models N$. Hence, $N$ is satisfiable, and so is $N_0$.

**Redundancy Elimination.** Based on the general redundancy notion for basic superposition, several effective *redundancy elimination rules* were presented in [14]. They allow deleting certain closures or replacing them with simpler ones in a derivation, without jeopardizing completeness. Next, we overview the most important redundancy elimination rules from [14].

A closure $C \cdot \sigma$ is *reduced modulo substitution $\eta$ relative to* a closure $D \cdot \theta$ if, for each rewrite systems $R$ and each ground substitution $\tau$, $C \cdot \sigma\eta\tau$ is variable irreducible w.r.t. $R$ whenever $D \cdot \theta\tau$ is variable irreducible w.r.t. $R$. Checking this condition is difficult, since one needs to consider all ground substitutions and all rewrite systems; however, approximate checks suitable for practice are known. One such check is based on the notion of *$\eta$-domination*: for two terms $s \cdot \sigma$ and $t \cdot \theta$, we say that $s$ is $\eta$-dominated by $t$, written $s \cdot \sigma \sqsubseteq_\eta t \cdot \theta$, if and only if $(i)$ $s\sigma\eta = t\theta$, and $(ii)$ whenever some variable $x$ from $\sigma$ occurs in $s$ at position $p$, then $p$ is in $t$ at or below a position of a variable.

For example, let $s \cdot \sigma = f(g(x), [g(y)])$ and $t \cdot \theta = f([g(c)], [g(h(z))])$. For a substitution $\eta = \{x \mapsto c, y \mapsto h(z)\}$, obviously $s\sigma\eta = t\theta$. Also, each marked position from $s \cdot \sigma$ can be overlaid at or inside a marked position of $t \cdot \theta$, so $s \cdot \sigma \sqsubseteq_\eta t \cdot \theta$.

This notion can be extended to literals as follows: $(s \approx t) \cdot \sigma \sqsubseteq_\eta (w \approx v) \cdot \theta$ if and only if $s \cdot \sigma \sqsubseteq_\eta w \cdot \theta$ and $t \cdot \sigma \sqsubseteq_\eta v \cdot \theta$, or $s \cdot \sigma \sqsubseteq_\eta v \cdot \theta$ and $t \cdot \sigma \sqsubseteq_\eta w \cdot \theta$. The definition is analogous for negative literals. Furthermore, a positive literal does not $\eta$-dominate a negative literal and vice versa. The extension to closures is performed as follows: $C \cdot \sigma \sqsubseteq_\eta D \cdot \theta$ if and only if, for each literal $L_1 \cdot \sigma$ from $C \cdot \sigma$, there exists a distinct literal $L_2 \cdot \theta$ from $D \cdot \theta$ such that $L_1 \cdot \sigma \sqsubseteq_\eta L_2 \cdot \theta$. Note that $D \cdot \theta$ is allowed to have more literals than $C \cdot \sigma$.

Now if $C \cdot \sigma \sqsubseteq_\eta D \cdot \theta$, then $C \cdot \sigma$ is reduced relative to $D \cdot \theta$ modulo $\eta$. For some $\eta$, it can happen that $L'\sigma\eta = L\theta$ holds, but $L' \cdot \sigma \sqsubseteq_\eta L \cdot \theta$ does not. Then, $L' \cdot \sigma$ can be made reduced relative to $L \cdot \theta$ by retracting those positions in $L \cdot \sigma$ that do not overlay into a substitution position of $L'$. Such a transformation enables an application of a simplification or deletion rule, while retracting as little information in $L' \cdot \sigma$ as possible.

A closure $C \cdot \sigma$ is a *basic subsumer* of $D \cdot \theta$ if there is a substitution $\eta$ such that $C\sigma\eta \subseteq D\theta$ and $C \cdot \sigma$ is reduced relative to $D \cdot \theta$ modulo $\eta$. Additionally, if $C \cdot \sigma$ has fewer literals than $D \cdot \theta$, then $D \cdot \theta$ can be deleted.

A closure $(C \vee A \vee B) \cdot \sigma$ can be replaced with $(C \vee A) \cdot \sigma$ if $A \cdot \sigma \sqsubseteq_{\{\}} B \cdot \sigma$; this rule is called *duplicate literal deletion*.

A closure $C \cdot \sigma$ can be deleted if $C\sigma$ is a tautology; this rule is called *tautology deletion*. Testing whether $C\sigma$ is a tautology itself requires theorem proving, so a semantic check is practically unfeasible. However, the following simple syntactic checks are effective in practice: $C \cdot \sigma$ is a *syntactic tautology* if it contains a pair of literals $(s \approx t) \cdot \sigma$ and $(s' \not\approx t') \cdot \sigma$ such that $s\sigma = s'\sigma$ and $t\sigma = t'\sigma$, or a literal of the form $(s \approx t) \cdot \sigma$ such that $s\sigma = t\sigma$.

A closure $(C \vee x \not\approx s) \cdot \sigma$ with $x\sigma \succ s\sigma$ is called a *basic tautology* and can be safely deleted. For example, if $f(x) \succ g(x)$, then the closure $[f(x)] \not\approx g(x)$ is a basic tautology. Note that $f(x) \not\approx g(x)$ is not a basic tautology, since $f(x)$ does not occur at a substitution position.

All presented redundancy elimination rules are decidable. In fact, duplicate literal deletion and tautology deletion can be performed in polynomial time. The subsumption check is NP-complete in the number of literals [53], and $\eta$-domination can be checked in polynomial time. The complexity of basic tautology deletion is determined by the complexity of checking ordering constraints. Finally, terms $s$ and $t$ can be compared by a lexicographic path ordering in time $\mathcal{O}(|s| \cdot |t|)$ [86, 138].

**Examples.** We now give several examples of $\mathcal{BS}$ inferences. We first consider a resolution inference, with an assumption that the parameters of $\mathcal{BS}$ make the literal $R(x, f(x))$ maximal in the first, and the literal $\neg R(x, y)$ selected in the second premise. The $\mathcal{E}$-terms that participate in an inference are denoted $\boxed{\text{like this}}$. The closure on the left-hand side is the side premise, whereas the closure on the right-hand side is the main premise. To apply the inference rule, we separate the variables in premises, compute the most general unifier $\sigma = \mathsf{MGU}(R(x, f(x)), R(x', y)) = \{x' \mapsto x, y \mapsto f(x)\}$, and apply it to the premises. By doing so, the literals on which the resolution takes place become

identical to $R(x, f(x))$, which allows the resolution to be performed. However, note that we actually apply $\sigma$ to the substitution part of the premises, so the substitution part effectively accumulates the terms introduced by unification. After resolution, the obtained closure is not in the standard form, since the substitution contains a trivial mapping $x' \mapsto x$; we bring the closure into standard form by applying the mapping to the skeleton and the substitution.

$$\dfrac{\begin{array}{cc} C(x) \vee \boxed{R(x,f(x))} \cdot \{\} & \neg D(x) \vee \boxed{\neg R(x,y)} \vee E(y) \cdot \{\} \\ \Downarrow & \Downarrow \\ C(x) \vee \boxed{R(x,f(x))} \cdot \{\} & \neg D(x') \vee \boxed{\neg R(x',y)} \vee E(y) \cdot \{\} \\ \Downarrow & \Downarrow \\ C(x) \vee \boxed{R(x,f(x))} \cdot \{\} & \neg D(x') \vee \boxed{\neg R(x',y)} \vee E(y) \cdot \{x' \mapsto x, y \mapsto f(x)\} \end{array}}{\begin{array}{c} C(x) \vee \neg D(x') \vee E(y) \cdot \{x' \mapsto x, y \mapsto f(x)\} \\ \Downarrow \\ C(x) \vee \neg D(x) \vee E(y) \cdot \{y \mapsto f(x)\} \end{array}}$$

The previous inference is written using the notation that explicitly distinguishes the skeleton from the substitution part of a closure. Next, we show the same inference written using the convenient notation, where terms occurring at positions of skeleton variables are marked. Note that, in the second step, the term $f(x)$ in the literal $E([f(x)])$ is introduced by unification and is therefore marked.

$$\dfrac{\begin{array}{cc} C(x) \vee \boxed{R(x,f(x))} & \neg D(x) \vee \boxed{\neg R(x,y)} \vee E(y) \\ \Downarrow & \Downarrow \\ C(x) \vee \boxed{R(x,f(x))} & \neg D(x') \vee \boxed{\neg R(x',y)} \vee E(y) \\ \Downarrow & \Downarrow \\ C(x) \vee \boxed{R(x,f(x))} & \neg D(x) \vee \boxed{\neg R(x,[f(x)])} \vee E([f(x)]) \end{array}}{C(x) \vee \neg D(x) \vee E([f(x)])}$$

Next, we give an example of a positive superposition inference. We assume that no literal in either premise is selected, that literals $y_1 \approx y_2 \cdot \{y_1 \mapsto f(x), y_2 \mapsto g(x)\}$ and $C(f(y)) \cdot \{y \mapsto f(x)\}$ are maximal, and that $f(x) \succ g(x)$. Superposition is performed from $y_1$ into $f(y)$. To apply the inference rule, we separate the variables in the premises, compute the most general unifier $\sigma = \mathsf{MGU}(f(x'), f(f(x))) = \{x' \mapsto f(x)\}$, and apply it to the premises. We then perform superposition, after which we remove from the substitution all mappings of variables that do not occur in the skeleton. Observe that the literal $C(f(y)) \cdot \{y \mapsto f(x)\}$ is equivalent to $C(f(f(x)))$, so one might attempt to perform superposition into inner $f(x)$. However, this is not allowed: the skeleton contains the variable $y$ at the position of inner $f(x)$, and, by superposition conditions, the term into which superposition is performed should not be a variable.

$$\dfrac{\begin{array}{cc} C(x) \vee \boxed{y_1} \approx y_2 \cdot \{y_1 \mapsto f(x), y_2 \mapsto g(x)\} & \neg D(x) \vee C(\boxed{f(y)}) \cdot \{y \mapsto f(x)\} \\ \Downarrow & \Downarrow \\ C(x') \vee \boxed{y_1} \approx y_2 \cdot \{y_1 \mapsto f(x'), y_2 \mapsto g(x')\} & \neg D(x) \vee C(\boxed{f(y)}) \cdot \{y \mapsto f(x)\} \\ \Downarrow & \Downarrow \\ C(x') \vee \boxed{y_1} \approx y_2 \cdot \{x' \mapsto f(x), y_1 \mapsto f(f(x)), y_2 \mapsto g(f(x))\} & \neg D(x) \vee C(\boxed{f(y)}) \cdot \{y \mapsto f(x)\} \end{array}}{\begin{array}{c} C(x') \vee \neg D(x) \vee C(y_2) \cdot \{x' \mapsto f(x), y_1 \mapsto f(f(x)), y_2 \mapsto g(f(x)), y \mapsto f(x)\} \\ \Downarrow \\ C(x') \vee \neg D(x) \vee C(y_2) \cdot \{x' \mapsto f(x), y_2 \mapsto g(f(x))\} \end{array}}$$

We now present the same inference using the convenient notation. That superposition into positions of skeleton variables is not allowed now means that superposition into terms which are under a marker is not allowed. For example, since the inner $f(x)$ in $E(f([f(x)]))$ is marked, superposition into it is not allowed.

$$\dfrac{\begin{array}{cc} C(x) \vee \boxed{[f(x)]} \approx [g(x)] & \neg D(x) \vee C(\boxed{f([f(x)])}) \\ \Downarrow & \Downarrow \\ C(x') \vee \boxed{[f(x')]} \approx [g(x')] & \neg D(x) \vee C(\boxed{f([f(x)])}) \\ \Downarrow & \Downarrow \\ C([f(x)]) \vee \boxed{[f(f(x))]} \approx [g(f(x))] & \neg D(x) \vee C(\boxed{f([f(x)])}) \end{array}}{C([f(x)]) \vee \neg D(x) \vee C([g(f(x))])}$$

We finish with an example of closure subsumption. Consider $C_1 = C(x) \vee D(f(x))$ and $C_2 = C([g(y)]) \vee D([f(g(y))]) \vee E(h(y))$. It is easy to see that $C_1$ subsumes $C_2$ by substitution $\eta = \{x \mapsto g(y)\}$: $(i)$ $C_1\eta = C([g(y)]) \vee D(f([g(y)]))$, so, by disregarding markers, $C_1\eta \subseteq C_2$, and $(ii)$ each marked subterm from $C_1\eta$ can be overlaid into a marked subterm in $C_2$. However, for $C_3 = C([g(y)]) \vee D(f(g(y))) \vee E(h(y))$, we can see that $C_1$ does not subsume $C_3$ by $\eta$: the marked subterm $[g(y)]$ from $D(f([g(y)]))$ cannot be overlaid into a marked term in $C_3$, since in the latter closure the term $g(y)$ in literal $D(f(g(y)))$ occurs unmarked. Actually, $C_1$ does not subsume $C_3$ under any substitution.

## 2.6 Splitting

In some proofs in the following chapters we use an additional splitting inference rule, which is borrowed from the semantic tableau calculus. If a closure consists of two parts not sharing common variables, one can separately assume that either part is true. If unsatisfiability is proved in both cases, the initial closure set is evidently unsatisfiable. Hence, splitting performs an explicit case analysis.

**Splitting:**
$$\frac{N \cup \{C \vee D\}}{N \cup \{C\} \mid N \cup \{D\}}$$

where $(i)$ $N$ is a set of closures, $(ii)$ $C$ and $D$ do not have variables in common.

Splitting changes the nature of resolution significantly: a derivation is now not unique, but is computed nondeterministically, and is called a *branch*. A set of closures $N$ is satisfiable if a branch exists that is saturated up to redundancy and does not contain the empty closure.

## 2.7 Disjunctive Datalog

The following presentation of the syntax and the semantics of disjunctive datalog is based on [42, 55]. Let $\Sigma$ be a first-order signature such that $(i)$ $\mathcal{F}(\Sigma)$ contains only constants, and $(ii)$ $\approx \in \mathcal{P}(\Sigma)$ is a special equality predicate with the arity of two. A *disjunctive datalog program with equality* $P$ is a finite set of rules of the form

$$A_1 \vee ... \vee A_n \leftarrow B_1, ..., B_m$$

where $n \geq 0$, $m \geq 0$, and $A_i$ and $B_i$ are atoms defined over $\Sigma$. Furthermore, each rule must be *safe*; that is, each variable occurring in a head literal must occur in a body literal as well. For a rule $r$, the set of atoms $\mathsf{head}(r) = \{A_i \,|\, 1 \leq i \leq n\}$ is called the *rule head*, whereas the set of atoms $\mathsf{body}(r) = \{B_i \,|\, 1 \leq i \leq m\}$ is called the *rule body*. A rule with an empty body is called a *fact*.

Typical definitions of a disjunctive datalog program, such as [42, 55], allow negated atoms in the body. This negation is usually nonmonotonic, and is thus different from negation in first-order logic. Our algorithms from the following chapters produce only positive disjunctive datalog programs, so we omit nonmonotonic negation from the definitions. Disjunctive datalog programs without negation-as-failure are often called *positive programs*.

The *ground instance* of $P$ over the Herbrand universe of $P$, written $\mathsf{ground}(P, HU)$, is the set of ground rules obtained by replacing all variables in each rule of $P$ with constants from $HU$ in all possible ways. The *Herbrand base* $HB$ of $P$ is the set of all ground atoms defined over predicates from $\mathcal{P}(\Sigma)$. An *interpretation* $M$ of $P$ is a subset of $HB$. An interpretation $M$ is a *model* of $P$ if the following conditions are satisfied: $(i)$ $\mathsf{body}(r) \subseteq M$ implies $\mathsf{head}(r) \cap M \neq \emptyset$, for each rule $r \in \mathsf{ground}(P, HU)$; and $(ii)$ all atoms from $M$ with the $\approx$ predicate yield a congruence relation—that is, a relation that is reflexive, symmetric, transitive, and $R(a_1, \ldots, a_i, \ldots, a_n) \in M$ and $a_i \approx b_i \in M$ imply $R(a_1, \ldots, b_i, \ldots, a_n) \in M$, for each predicate symbol $R \in \mathcal{P}(\Sigma)$.

A model $M$ of $P$ is *minimal* if no subset of $M$ is a model of $P$. The semantics of $P$ is defined as the set of all minimal models of $P$, denoted by $\mathcal{MM}(P)$. Finally, the notion of query answering is defined as follows. A ground literal $A$ is a *cautious answer* of $P$, written $P \models_c A$, if $A \in M$ for all $M \in \mathcal{MM}(P)$; $A$ is a *brave answer* of $P$, written $P \models_b A$, if $A \in M$ for at least one $M \in \mathcal{MM}(P)$. First-order entailment coincides with cautious entailment for positive ground atoms on positive programs.

The size of a rule $r$ is defined as $|r| = 1 + \sum_{1 \leq i \leq n} |A_i| + \sum_{1 \leq j \leq m} |B_j|$, where the size of atoms $A_i$ and $B_j$ is defined as $|S(t_1, \ldots, t_n)| = 1 + n$: predicates and terms are encoded with one symbol, and the leading 1 in the definition of $|r|$ accounts for the implication symbol separating the head from the body. The size of a program $P$, written $|P|$, is the sum of the sizes of all its rules.

# Chapter 3

# Introduction to Description Logics

In this chapter, we present a formal definition of the syntax and the semantics of description logics, as well as of the interesting inference problems. We introduce the basic description logic $\mathcal{SHIQ}$ in Section 3.1, and extend it to $\mathcal{SHIQ}(\mathbf{D})$ in Section 3.2 by adding *datatypes*. We also give examples of $\mathcal{SHIQ}$ and $\mathcal{SHIQ}(\mathbf{D})$ knowledge bases, and of interesting conclusions that can be drawn from them. We use these examples in the latter chapters to demonstrate our reasoning algorithms.

## 3.1 The Description Logic $\mathcal{SHIQ}$

The syntax of the description logic $\mathcal{SHIQ}$ [73] is defined as follows.

**Definition 3.1.1.** *For a set of abstract role names $N_{R_a}$, the set of $\mathcal{SHIQ}$* abstract roles *is $N_{R_a} \cup \{R^- \mid R \in N_{R_a}\}$. Let $\mathsf{Inv}(R) = R^-$ and $\mathsf{Inv}(R^-) = R$ for $R \in N_{R_a}$. A $\mathcal{SHIQ}$* RBox *$KB_{\mathcal{R}}$ over $N_{R_a}$ is a finite set of transitivity axioms $\mathsf{Trans}(R)$ and abstract role inclusion axioms $R \sqsubseteq S$ such that $R \sqsubseteq S \in KB_{\mathcal{R}}$ implies $\mathsf{Inv}(R) \sqsubseteq \mathsf{Inv}(S) \in KB_{\mathcal{R}}$, and $\mathsf{Trans}(R) \in KB_{\mathcal{R}}$ implies $\mathsf{Trans}(\mathsf{Inv}(R)) \in KB_{\mathcal{R}}$.*

*Let $\sqsubseteq^*$ be the reflexive–transitive closure of $\sqsubseteq$. A role $R$ is* transitive *if there is a role $S$ such that $\mathsf{Trans}(S) \in \mathcal{R}$ with $S \sqsubseteq^* R$ and $R \sqsubseteq^* S$; $R$ is* simple *if there is no role $S$ such that $S \sqsubseteq^* R$ and $S$ is transitive; and $R$ is* complex *if it is not simple.*

*Let $N_C$ be a set of* atomic *concepts. The set of $\mathcal{SHIQ}$* concepts *over $N_C$ and $N_{R_a}$ is defined inductively as the minimal set for which the following holds: $\top$ and $\bot$ are $\mathcal{SHIQ}$ concepts; each atomic concept $A \in N_C$ is a $\mathcal{SHIQ}$ concept; and, if $C$ and $D$ are $\mathcal{SHIQ}$ concepts, $R$ is an abstract role, $S$ is an abstract simple role, and $n$ is an integer, then $\neg C$, $C \sqcap D$, $C \sqcup D$, $\exists R.C$, $\forall R.C$, $\leq n\, S.C$, and $\geq n\, S.C$ are also $\mathcal{SHIQ}$ concepts. Concepts that are not in $N_C$ are called* complex*. Possibly negated atomic concepts are called* literal concepts*. A concept $C$ is a* subconcept *of a concept $D$ if $C$ syntactically occurs in $D$.*

*A $\mathcal{SHIQ}$* TBox *$KB_{\mathcal{T}}$ over $N_C$ and $KB_{\mathcal{R}}$ is a finite set of* concept inclusion axioms *$C \sqsubseteq D$ or* concept equivalence axioms *$C \equiv D$, where $C$ and $D$ are $\mathcal{SHIQ}$ concepts.*

*Let $N_{I_a}$ be a set of* abstract individuals. *A $\mathcal{SHIQ}$* ABox *$KB_{\mathcal{A}}$ is a set of concept and abstract role* membership axioms *$C(a)$, $R(a,b)$, $\neg S(a,b)$, and* (in)equality axioms *$a \approx b$ and $a \not\approx b$, where $C$ is a $\mathcal{SHIQ}$ concept, $R$ is an abstract role, $S$ is an abstract simple role, and $a$ and $b$ are abstract individuals. An ABox is* extensionally reduced *if all ABox axioms contain only literal concepts.*

*A $\mathcal{SHIQ}$ knowledge base $KB$ is a triple $(KB_{\mathcal{R}}, KB_{\mathcal{T}}, KB_{\mathcal{A}})$, where $KB_{\mathcal{R}}$ is an RBox, $KB_{\mathcal{T}}$ is a TBox, $KB_{\mathcal{A}}$ is an ABox, and where the sets $N_{R_a}$, $N_C$, and $N_{I_a}$ are mutually disjoint.*

Definition 3.1.1 differs from typical definitions in two aspects. First, OWL-DL lacks the unique name assumption (UNA), so we do not incorporate UNA into the definition of $\mathcal{SHIQ}$, but allow the user to axiomatize it by including an inequality axiom $a_i \not\approx a_j$ for each pair of distinct abstract individuals [4, page 60]. Second, usual definitions do not provide for ABox axioms involving negative roles. We allow such assertions, because they allow checking entailment of ground role facts. Third, it would be possible to allow complex roles in negative role membership axioms. However, our approach for dealing with transitivity from Section 5.2 cannot handle such axioms, so we adopt this weaker definition.

**Definition 3.1.2.** *The semantics of a $\mathcal{SHIQ}$ knowledge base $KB$ is given by the mapping $\pi$ that transforms $KB$ axioms into a first-order formula, as shown in Table 3.1. An atomic concept is mapped into a unary predicate, an abstract role is mapped into a binary predicate, and an abstract individual is mapped into a constant.*

*The basic inference problem for $\mathcal{SHIQ}$ is checking* satisfiability *of $KB$—that is, determining whether a first-order model of $\pi(KB)$ exists. Other interesting inference problems can be reduced to satisfiability as follows, where $\iota$ is a new abstract individual not occurring in the knowledge base:*

- Concept satisfiability: *A concept $C$ is* satisfiable *with respect to $KB$ if and only if there exists a model of $KB$ in which the interpretation of $C$ is not empty. This is the case if and only if $KB \cup \{C(\iota)\}$ is satisfiable.*
- Subsumption: *A concept $C$ is* subsumed by *a concept $D$ with respect to $KB$, written $KB \models C \sqsubseteq D$, if and only if $\pi(KB) \models \pi(C \sqsubseteq D)$. This is the case if and only if $KB \cup \{(C \sqcap \neg D)(\iota)\}$ is unsatisfiable.*
- Concept equivalence: *A concept $C$ is* equivalent to *a concept $D$ with respect to $KB$, written $KB \models C \equiv D$, if and only if $C$ subsumes $D$ with respect to $KB$ and vice versa.*
- Instance checking: *An individual $a$ is an* instance of *a concept $C$ with respect to $KB$, written $KB \models C(a)$, if and only if $\pi(KB) \models \pi(C(a))$. This is the case if and only if $KB \cup \{\neg C(a)\}$ is unsatisfiable.*

- Role checking: *A simple abstract role $S$* relates *abstract individuals $a$ and $b$ with respect to $KB$, written $KB \models R(a,b)$, if and only if $\pi(KB) \models \pi(S(a,b))$. This is the case if and only if $KB \cup \{\neg S(a,b)\}$ is unsatisfiable.*

The semantics of description logics is usually given by a direct model-theoretic semantics. An interpretation $I = (\triangle^I, \cdot^I)$ consists of a nonempty domain set $\triangle^I$ and an interpretation function $\cdot^I$ that assigns an element $a^I \in \triangle^I$ to each individual $a$, a set $A^I \subseteq \triangle^I$ to each atomic concept $A$, and a relation $R^I \subseteq \triangle^I \times \triangle^I$ to each role $R$. The semantics of complex concepts and axioms is given in Table 3.2, where $C$ and $D$ are concepts, $R$ and $S$ are roles, $a$ and $b$ are individuals, and $\sharp N$ is the number of elements in a set $N$. The direct model-theoretic semantics and the semantics by translation into first-order logic coincide, as first shown by Borgida [21]. For a role $R$ and an object $x \in \triangle^I$, an object $y \in \triangle^I$ is called an *$R$-successor of $x$ in $I$* if $(x,y) \in R^I$.

Often, we need a way to compare the expressivity of different description logics. The following definition provides means for that by a simple syntactic comparison.

**Definition 3.1.3.** *Let $\mathcal{L}$, $\mathcal{L}_1$, and $\mathcal{L}_2$ be three description logics.*

- *The logic $\mathcal{L}$ is a* fragment *of $\mathcal{L}_1$ if each axiom of $\mathcal{L}$ is also an axiom of $\mathcal{L}_1$.*
- *The logic $\mathcal{L}$ is* between *$\mathcal{L}_1$ and $\mathcal{L}_2$ if $\mathcal{L}_1$ is a fragment of $\mathcal{L}$, and $\mathcal{L}$ is a fragment of $\mathcal{L}_2$.*

We now define several fragments of $\mathcal{SHIQ}$, which we use in the following chapters.

**Definition 3.1.4.** *For a knowledge base $KB$, a role $R$ is called* very simple *if no role $S$ exists such that $S \sqsubseteq R \in KB_{\mathcal{R}}$. The description logic $\mathcal{SHIQ}^-$ is a fragment of $\mathcal{SHIQ}$, where only very simple roles are allowed to occur in number restrictions $\leq n\, R.C$ and $\geq n\, R.C$.*

*$\mathcal{ALCHIQ}$ ($\mathcal{ALCHIQ}^-$) is a fragment of $\mathcal{SHIQ}$ ($\mathcal{SHIQ}^-$) that does not allow transitivity axioms in RBoxes. $\mathcal{ALC}$ is the fragment of $\mathcal{ALCHIQ}$ that does not provide for role inclusion axioms, inverse roles, and number restrictions.*

Observe that, if $KB$ is not extensionally reduced, it can be easily transformed into an extensionally reduced knowledge base: for each axiom $C(a)$ where $C$ is not a literal concept, one can introduce a new atomic concept $A_C$, add the axiom $A_C \sqsubseteq C$ to the TBox, and replace $C(a)$ with $A_C(a)$. Such a transformation is obviously polynomial in the number of individuals, so, without loss of generality, it is safe to assume that a knowledge base is extensionally reduced.

Note that, by Definition 3.1.1, the relation $\sqsubseteq^*$ can be cyclic in general. In [144] it was shown that we can reduce each $\mathcal{SHIQ}$ knowledge base $KB$ with a cyclic role hierarchy to a $\mathcal{SHIQ}$ knowledge base $KB'$ with an acyclic role hierarchy using the following algorithm. First, we compute the set of maximal, strongly connected components (or maximal cycles) of the role inclusion relation $\sqsubseteq$ of $KB$. For each strongly connected component $\Gamma$, we select one representative role, denoted with $\mathsf{role}(\Gamma)$, such

Table 3.1: Semantics of $\mathcal{SHIQ}$ by Mapping to FOL

| | | |
|---|---|---|
| **Mapping Concepts to FOL** | | |
| $\pi_y(\top, X)$ | $=$ | $\top$ |
| $\pi_y(\bot, X)$ | $=$ | $\bot$ |
| $\pi_y(A, X)$ | $=$ | $A(X)$ |
| $\pi_y(\neg C, X)$ | $=$ | $\neg\pi_y(C, X)$ |
| $\pi_y(C \sqcap D, X)$ | $=$ | $\pi_y(C, X) \wedge \pi_y(D, X)$ |
| $\pi_y(C \sqcup D, X)$ | $=$ | $\pi_y(C, X) \vee \pi_y(D, X)$ |
| $\pi_y(\forall R.C, X)$ | $=$ | $\forall y : R(X, y) \rightarrow \pi_x(C, y)$ |
| $\pi_y(\exists R.C, X)$ | $=$ | $\exists y : R(X, y) \wedge \pi_x(C, y)$ |
| $\pi_y(\leq n\, R.C, X)$ | $=$ | $\forall y_1, \ldots, y_{n+1} : \bigwedge_{i=1}^{n+1}[R(X, y_i) \wedge \pi_x(C, y_i)] \rightarrow \bigvee_{i=1}^{n+1} \bigvee_{j=i+1}^{n+1} y_i \approx y_j$ |
| $\pi_y(\geq n\, R.C, X)$ | $=$ | $\exists y_1, \ldots, y_n : \bigwedge_{i=1}^{n}[R(X, y_i) \wedge \pi_x(C, y_i)] \wedge \bigwedge_{i=1}^{n} \bigwedge_{j=i+1}^{n} y_i \not\approx y_j$ |
| **Mapping Axioms to FOL** | | |
| $\pi(C \sqsubseteq D)$ | $=$ | $\forall x : \pi_y(C, x) \rightarrow \pi_y(D, x)$ |
| $\pi(C \equiv D)$ | $=$ | $\forall x : \pi_y(C, x) \leftrightarrow \pi_y(D, x)$ |
| $\pi(R \sqsubseteq S)$ | $=$ | $\forall x, y : R(x, y) \rightarrow S(x, y)$ |
| $\pi(\mathsf{Trans}(R))$ | $=$ | $\forall x, y, z : R(x, y) \wedge R(y, z) \rightarrow R(x, z)$ |
| $\pi(C(a))$ | $=$ | $\pi_y(C, a)$ |
| $\pi(R(a, b))$ | $=$ | $R(a, b)$ |
| $\pi(\neg S(a, b))$ | $=$ | $\neg S(a, b)$ |
| $\pi(a \circ b)$ | $=$ | $a \circ b$ for $\circ \in \{\approx, \not\approx\}$ |
| **Mapping $KB$ to FOL** | | |
| $\pi(R)$ | $=$ | $\forall x, y : R(x, y) \leftrightarrow R^{-}(y, x)$ |
| $\pi(KB_{\mathcal{R}})$ | $=$ | $\bigwedge_{\alpha \in KB_{\mathcal{R}}} \pi(\alpha) \wedge \bigwedge_{R \in N_{R_a}} \pi(R)$ |
| $\pi(KB_{\mathcal{T}})$ | $=$ | $\bigwedge_{\alpha \in KB_{\mathcal{T}}} \pi(\alpha)$ |
| $\pi(KB_{\mathcal{A}})$ | $=$ | $\bigwedge_{\alpha \in KB_{\mathcal{A}}} \pi(\alpha)$ |
| $\pi(KB)$ | $=$ | $\pi(KB_{\mathcal{R}}) \wedge \pi(KB_{\mathcal{T}}) \wedge \pi(KB_{\mathcal{A}})$ |

Notes:
($i$): $X$ is a meta-variable and is substituted by the actual term;
($ii$): $\pi_x$ is obtained from $\pi_y$ by simultaneously substituting in the definition all $y_{(i)}$ with $x_{(i)}$, $\pi_y$ with $\pi_x$, and vice versa.

Table 3.2: Direct Model-Theoretic Semantics of $\mathcal{SHIQ}$

| | | |
|---|---|---|
| Interpreting Concepts | | |
| $\top^I$ | $=$ | $\triangle^I$ |
| $\bot^I$ | $=$ | $\emptyset$ |
| $(\neg C)^I$ | $=$ | $\triangle^I \setminus C^I$ |
| $(C \sqcap D)^I$ | $=$ | $C^I \cap D^I$ |
| $(C \sqcup D)^I$ | $=$ | $C^I \cup D^I$ |
| $(\forall R.C)^I$ | $=$ | $\{x \mid \forall y : (x,y) \in R^I \rightarrow y \in C^I\}$ |
| $(\exists R.C)^I$ | $=$ | $\{x \mid \exists y : (x,y) \in R^I \wedge y \in C^I\}$ |
| $(\leq n\, R.C)^I$ | $=$ | $\{x \mid \sharp\{y \mid (x,y) \in R^I \wedge y \in C^I\} \leq n\}$ |
| $(\geq n\, R.C)^I$ | $=$ | $\{x \mid \sharp\{y \mid (x,y) \in R^I \wedge y \in C^I\} \geq n\}$ |
| Semantics of Axioms | | |
| $C \sqsubseteq D$ | | $C^I \subseteq D^I$ |
| $C \equiv D$ | | $C^I = D^I$ |
| $R \sqsubseteq S$ | | $R^I \subseteq S^I$ |
| $\mathsf{Trans}(R)$ | | $(R^I)^+ \subseteq R^I$ |
| $C(a)$ | | $a^I \in C^I$ |
| $R(a,b)$ | | $(a^I, b^I) \in R^I$ |
| $\neg S(a,b)$ | | $(a^I, b^I) \notin S^I$ |
| $a \approx b$ | | $a^I = b^I$ |
| $a \not\approx b$ | | $a^I \neq b^I$ |

that, if $R \in \Gamma$ and $\mathsf{Inv}(R) \in \Gamma'$ (where $\Gamma'$ is a strongly connected component possibly different from $\Gamma$) and $\mathsf{role}(\Gamma) = R$, then $\mathsf{role}(\Gamma') = \mathsf{Inv}(R)$. Since we assume that $R \sqsubseteq S \in KB_{\mathcal{R}}$ implies $\mathsf{Inv}(R) \sqsubseteq \mathsf{Inv}(S) \in KB_{\mathcal{R}}$, we have that, if $R, S \in \Gamma$ and $\mathsf{Inv}(R) \in \Gamma'$, then $\mathsf{Inv}(S) \in \Gamma'$, so the definition of $\mathsf{role}(\Gamma)$ is correct. Next, we form the new TBox $KB'_{\mathcal{T}}$ and ABox $KB'_{\mathcal{A}}$ by replacing, in all axioms of $KB_{\mathcal{A}}$ and $KB_{\mathcal{T}}$, each role $R$ with $\mathsf{role}(\Gamma)$, where $\Gamma$ is the maximal, strongly connected component that $R$ belongs to. Finally, we construct the new RBox $KB'_{\mathcal{R}}$ as follows. For each pair of strongly connected components $\Gamma \neq \Gamma'$, if there are roles $R \in \Gamma$ and $R' \in \Gamma'$ with $R \sqsubseteq R'$, we add the axiom $\mathsf{role}(\Gamma) \sqsubseteq \mathsf{role}(\Gamma')$ to $KB'_{\mathcal{R}}$. Next, for each strongly connected component $\Gamma$, we add the axiom $\mathsf{Inv}(\mathsf{role}(\Gamma)) \sqsubseteq \mathsf{role}(\Gamma)$ to $KB'_{\mathcal{R}}$ if there is a role $R \in \Gamma$ such that $\mathsf{Inv}(R) \in \Gamma$. Since the strongly connected components of $\sqsubseteq$ can be computed in time quadratic in the number of roles, this reduction can be performed in polynomial time. Hence, we can assume without loss of generality that $\sqsubseteq^*$ is acyclic.

A concept $C$ is in *negation-normal form* if all negations in $C$ occur in front of atomic concepts only. A concept $C$ can be transformed in time linear in the size of $C$ into an equivalent concept in negation-normal form, written $\mathsf{NNF}(C)$, by exhaustively applying the following rewrite rules to subconcepts of $C$:

$$\begin{array}{rcl@{\qquad}rcl}
\neg\top & \rightsquigarrow & \bot & \neg\bot & \rightsquigarrow & \top \\
\neg(C_1 \sqcap C_2) & \rightsquigarrow & \neg C_1 \sqcup \neg C_2 & \neg(C_1 \sqcup C_2) & \rightsquigarrow & \neg C_1 \sqcap \neg C_2 \\
\neg(\exists R.C) & \rightsquigarrow & \forall R.\neg C & \neg(\forall R.C) & \rightsquigarrow & \exists R.\neg C \\
\neg(\geq (n+1)\, R.C) & \rightsquigarrow & \leq n\, R.C & \neg(\leq n\, R.C) & \rightsquigarrow & \geq (n+1)\, R.C \\
\neg(\geq 0\, R.C) & \rightsquigarrow & \bot
\end{array}$$

With $|KB|$ we denote the *size of the knowledge base* assuming unary coding of numbers, which is computed recursively in the following way, for $C$ and $D$ concepts, $A$ an atomic concept, and $R$ and $S$ roles:

$$\begin{array}{rcl@{\qquad}rcl}
|\top| & = & 1 & |\bot| & = & 1 \\
|A| & = & 1 & |\neg C| & = & 1 + |C| \\
|C \sqcup D| & = & |C| + |D| + 1 & |C \sqcap D| & = & |C| + |D| + 1 \\
|\exists R.C| & = & 2 + |C| & |\forall R.C| & = & 2 + |C| \\
|\geq n\, R.C| & = & n + 2 + |C| & |\leq n\, R.C| & = & n + 2 + |C| \\
|C \sqsubseteq D| & = & |C| + |D| + 1 & |C \equiv D| & = & |C| + |D| + 1 \\
|R \sqsubseteq S| & = & 3 & |\mathsf{Trans}(R)| & = & 2 \\
|C(a)| & = & |C| + 1 & |R(a,b)| & = & 3 \\
|KB_{\mathcal{R}}| & = & \sum_{\alpha \in KB_{\mathcal{R}}} |\alpha| & |KB_{\mathcal{T}}| & = & \sum_{\alpha \in KB_{\mathcal{T}}} |\alpha| \\
|KB_{\mathcal{A}}| & = & \sum_{\alpha \in KB_{\mathcal{A}}} |\alpha| & |KB| & = & |KB_{\mathcal{R}}| + |KB_{\mathcal{T}}| + |KB_{\mathcal{A}}|
\end{array}$$

Intuitively, $|KB|$ is the number of symbols needed to encode $KB$ on the input tape of a Turing machine. We use a single symbol for each atomic concept, role, and individual. The $n$ in the definition of the length of concepts $\geq n\, R.C$ and $\leq n\, R.C$ stems from the assumption on number coding: a number $n$ can be encoded in unary with $n$ bits.

The notion of positions is extended to $\mathcal{SHIQ}$ concepts and axioms in the obvious way:

- $\alpha|_\epsilon = \alpha$ for $\alpha$ a concept or an axiom;
- $(\neg D)|_{1.p} = D|_p$;
- $(D_1 \circ D_2)|_{i.p} = D_i|_p$ for $\circ \in \{\sqcap, \sqcup, \sqsubseteq, \equiv\}$ and $i \in \{1, 2\}$;
- For $C = \bowtie R.D$ with $\bowtie \in \{\exists, \forall\}$, $C|_1 = R$, and $C|_{2.p} = D|_p$;
- For $C = \bowtie n\, R.D$ with $\bowtie \in \{\leq, \geq\}$, $C|_1 = n$, $C|_2 = R$, and $C|_{3.p} = D|_p$;
- $\mathsf{Trans}(R)|_1 = R$;
- For $\alpha = C(a)$, $\alpha|_{1.p} = C|_p$, and $\alpha|_2 = a$;
- For $\alpha = R(a, b)$, $\alpha|_1 = R$, $\alpha|_2 = a$, and $\alpha|_3 = b$;
- $(\neg S(a, b))|_{1.p} = S(a, b)|_p$;
- $(a_1 \circ a_2)|_i = a_i$ for $\circ \in \{\approx, \not\approx\}$ and $i \in \{1, 2\}$.

For $\alpha$ a $\mathcal{SHIQ}$ concept or axiom and $p$ a position in $\alpha$, replacing $\alpha|_p$ with $\beta$ is denoted with $\alpha[\beta]_p$, and is defined in the obvious way (we assume that the result is a syntactically correct term). Furthermore, if $\alpha|_p$ is a concept, then the *polarity* of $p$ is defined to agree with the polarity of the corresponding position in translation of $\alpha$ into first-order logic, and is defined as follows:

$$
\begin{aligned}
\mathsf{pol}(C, \epsilon) &= 1 \\
\mathsf{pol}(C_1 \sqcap C_2, i.p) &= \mathsf{pol}(C_i, p) \text{ for } i \in \{1,2\} \\
\mathsf{pol}(C_1 \sqcup C_2, i.p) &= \mathsf{pol}(C_i, p) \text{ for } i \in \{1,2\} \\
\mathsf{pol}(\leq n\, R.C, 3.p) &= -\mathsf{pol}(C, p) \\
\mathsf{pol}(C_1 \sqsubseteq C_2, 1.p) &= -\mathsf{pol}(C_1, p) \\
\mathsf{pol}(C_1 \equiv C_2, i.p) &= 0 \text{ for } i \in \{1,2\}
\end{aligned}
\qquad
\begin{aligned}
\mathsf{pol}(\neg C, 1.p) &= -\mathsf{pol}(C, p) \\
\mathsf{pol}(\exists R.C, 2.p) &= \mathsf{pol}(C, p) \\
\mathsf{pol}(\forall R.C, 2.p) &= \mathsf{pol}(C, p) \\
\mathsf{pol}(\geq n\, R.C, 3.p) &= \mathsf{pol}(C, p) \\
\mathsf{pol}(C_1 \sqsubseteq C_2, 2.p) &= \mathsf{pol}(C_2, p) \\
\mathsf{pol}(C(a), 1.p) &= \mathsf{pol}(C, p)
\end{aligned}
$$

### 3.1.1 Example

In this subsection, we give an example of a $\mathcal{SHIQ}$ knowledge base and show some inferences that can be drawn from it. DLs were often used for solving configuration problems [93]. Configurations are usually consist of many different parts, catalogs of parts that configurations are assembled from are usually hierarchical, and valid configurations can, to a large degree, be described using logical assertions. Hence, we embed our example in a configuration scenario.

Let ACME be a computer manufacturing company. As all modern suppliers of computer equipment, ACME allows its customers to assemble a computer configuration that best suits their needs. However, assembling a configuration is a nontrivial task, due to interdependencies among different components. For example, the choice of the motherboard usually constrains the types of compatible memory chips, disk controllers, and other components. To aid its customers in assembling a valid configuration, ACME creates a formal ontology of catalog parts. This ontology can be used in an application that guides the users in choosing their components, thus ensuring that chosen components work together correctly in an assembled computer.

To represent objects from the ACME catalog and the relationships between them, we use the concepts and roles from Table 3.3.

We now describe some relations that hold for all computers in all configurations. First, we would like to constrain the role *hasAdpt* to point only to instances of the *Adpt* concept; we often say that the *range* of the *hasAdpt* role is the *Adpt* concept. Similarly, the range of *has3DAcc* is *3DAcc*. These constraints can be specified using these axioms:

$$\top \sqsubseteq \forall\, \mathit{hasAdpt}.\mathit{Adpt} \tag{3.1}$$

$$\top \sqsubseteq \forall\, \mathit{has3DAcc}.\mathit{3DAcc} \tag{3.2}$$

Next, we specify taxonomical relationships that hold between items in the catalog. Axiom (3.3) states that things that are adapters and 3D accelerators are adapters with built-in 3D accelerators. Furthermore, (3.4) and (3.5) state that adapters and 3D

| Concepts | |
|---|---|
| *PC* | all personal computers |
| *GrWS* | all graphics workstations |
| *GaPC* | all computers mainly used for playing computer games |
| *Adpt* | all video adapters in the catalog |
| *3DAcc* | all 3D video accelerators |
| *Adpt3DAcc* | all video adapters with built-in 3D accelerators |
| *VD* | all video devices |
| *PCI* | all devices built for the PCI bus |
| **Roles** | |
| *hasAdpt* | computer has a video adapter |
| *has3DAcc* | computer has a 3D accelerator |
| *hasVD* | computer has a video device |
| *contains* | some object contains other objects |

Table 3.3: Example Terminology Used in ACME's Catalog

accelerators are video devices, respectively. Finally, (3.6) states that graphic workstations are PCs, and (3.7) states that gaming PCs are PCs and graphics workstations. (Namely, to play modern computer games, one usually needs more graphics power than for many business applications of computer graphics.)

$$\mathit{Adpt} \sqcap \mathit{3DAcc} \sqsubseteq \mathit{Adpt3DAcc} \tag{3.3}$$

$$\mathit{Adpt} \sqsubseteq \mathit{VD} \tag{3.4}$$

$$\mathit{3DAcc} \sqsubseteq \mathit{VD} \tag{3.5}$$

$$\mathit{GrWS} \sqsubseteq \mathit{PC} \tag{3.6}$$

$$\mathit{GaPC} \sqsubseteq \mathit{GrWS} \sqcap \mathit{PC} \tag{3.7}$$

Next, we specify relationships among roles. Axiom (3.8) states that "having a video adapter" is a kind of "having a video device," and (3.9) states that "having a 3D accelerator" is also a kind of "having a video device." Finally, "having a video device" is a kind of containment relationship, as specified by (3.10); the latter relationship is transitive by (3.11).

$$\mathit{hasAdpt} \sqsubseteq \mathit{hasVD} \tag{3.8}$$

$$\mathit{has3DAcc} \sqsubseteq \mathit{hasVD} \tag{3.9}$$

$$\mathit{hasVD} \sqsubseteq \mathit{contains} \tag{3.10}$$

$$\mathsf{Trans}(\mathit{contains}) \tag{3.11}$$

Finally, we specify what permitted configurations look like. A personal computer usually requires at least one video adapter, as stated by (3.12). Intuitively, one might expect the right-hand side of the axiom to be $\exists\, \mathit{hasAdpt}.\mathit{Adpt}$ instead of $\exists\, \mathit{hasAdpt}.\top$.

However, since the range of *hasAdpt* is *Adpt* by (3.1), the axiom (3.12) is sufficient. Next, (3.13) states that graphics workstations require a 3D accelerator. Axiom (3.14) states that gaming PCs, being destined for the lower market segment, should have at most one video device. Finally, (3.15) ensures that all parts of a PCI-based computer are really built for the PCI bus.

$$PC \sqsubseteq \exists\, hasAdpt.\top \tag{3.12}$$

$$GrWS \sqsubseteq \exists\, has3DAcc.\top \tag{3.13}$$

$$GaPC \sqsubseteq \leq 1\, hasVD.VD \tag{3.14}$$

$$PCI \sqsubseteq \forall\, contains.PCI \tag{3.15}$$

With $KB_1$ we denote a $\mathcal{SHIQ}$ knowledge base that contains exactly the axioms (3.1)–(3.15).

We now show how $KB_1$ can be used to solve configuration tasks. Let us now assume that a customer wants to buy a gaming PC, and that an ACME sales representative is trying to help the customer to decide what kind of video or 3D accelerator should be added to the computer. He represents the choices of the customer using the following assertion:

$$GaPC(pc_1) \tag{3.16}$$

Axiom (3.16) gives a name, $pc_1$, to the computer that the customer wants to buy, and states that it is a gaming PC; let $KB_2$ be a $\mathcal{SHIQ}$ ABox containing only the axiom (3.16).

The knowledge about the configuration and the user's choices can now be used to guide the construction of a valid configuration as follows. Namely, it is easy to see that the following holds:

$$KB_1 \cup KB_2 \models \exists hasVD.Adpt3DAcc(pc_1) \tag{3.17}$$

In other words, the video device that the customer needs must be a video adapter with a built-in 3D accelerator. Namely, since $pc_1$ is a gaming PC by (3.16), it has a video adapter by (3.12) and a 3D accelerator by (3.13). However, all gaming PCs are allowed to have only one video device by (3.14), so the video adapter and the 3D accelerator must be one and the same. Hence, the video device must be the video adaptor and the 3D accelerator.

## 3.2 Description Logics with Concrete Domains

Practical applications of description logics often need to represent *concrete* properties, such as height, name, or age, which assume values from a fixed domain, such as integers or strings. This requirement led to the extension of description logics with *concrete domains* [5]. Informally, a concrete domain provides a set of predicates with

a predefined interpretation. If a decision procedure for checking satisfiability of finite conjunctions over concrete domain predicates exists, many DLs can be combined with a concrete domain while retaining decidability. Unfortunately, in [89] it was shown that a logic with general inclusion axioms and concrete domains is undecidable. Therefore, in [70, 101, 62] several restrictions of the general approach were investigated; a survey of the main results was presented in [88]. The cumulative results of this research have influenced the design of OWL-DL [106], which supports *datatypes*—a basic form of concrete domains.

### 3.2.1 Concrete Domains

With $\mathbf{x}$ we denote a vector of variables $x_1, \ldots, x_n$, and for a function $\delta$, with $\delta(\mathbf{x})$ we denote the application of $\delta$ to each element $x_i$ of $\mathbf{x}$—that is, a vector $\delta(x_1), \ldots, \delta(x_n)$. A concrete domain is defined as follows:

**Definition 3.2.1.** *A* concrete domain $\mathbf{D}$ *is a pair* $(\triangle_{\mathbf{D}}, \Phi_{\mathbf{D}})$*, where* $\triangle_{\mathbf{D}}$ *is a set, called the* domain *of* $\mathbf{D}$*, and* $\Phi_{\mathbf{D}}$ *is a finite set of* concrete predicate names*. Each* $d \in \Phi_{\mathbf{D}}$ *is associated with an arity* $n$ *and an* extension $d^{\mathbf{D}} \subseteq \triangle_{\mathbf{D}}^n$*. A concrete domain* $\mathbf{D}$ *is* admissible *if the following conditions hold:*

- $\Phi_{\mathbf{D}}$ *is closed under negation; that is, for each* $d \in \Phi_{\mathbf{D}}$*, there exists* $\overline{d} \in \Phi_{\mathbf{D}}$ *with* $\overline{d}^{\mathbf{D}} = \triangle_{\mathbf{D}}^n \setminus d^{\mathbf{D}}$*;*
- $\Phi_{\mathbf{D}}$ *contains a unary predicate* $\top_{\mathbf{D}}$ *interpreted as* $\triangle_{\mathbf{D}}$*;*
- $\Phi_{\mathbf{D}}$ *contains a binary predicate* $\approx_{\mathbf{D}}$ *interpreted as* $\{(x, y) \mid x = y\}$*;*
- $\Phi_{\mathbf{D}}$ *contains a unary predicate* $=_\alpha$ *for each* $\alpha \in \triangle_{\mathbf{D}}$*, interpreted as* $\{\alpha\}$*;*
- $\mathbf{D}$*-satisfiability of finite conjunctions of the form* $\bigwedge_{i=1}^n d_i(\mathbf{x_i})$*—that is, checking if an assignment* $\delta$ *of variables to elements of* $\triangle_{\mathbf{D}}$ *exists such that* $\delta(\mathbf{x_i}) \in d_i^{\mathbf{D}}$ *for each* $1 \leq i \leq n$*—is decidable.*

The definition of admissibility from [5] does not require the existence of $\approx_{\mathbf{D}}$ predicate. However, the algorithms we develop in the subsequent chapters depend on this predicate, so we extend the notion of admissibility appropriately. Furthermore, the usual definition does not require the existence of predicates $=_\alpha$. We introduce this restriction to allow accessing concrete domain individuals explicitly. Thus, we can state that the age of Peter is 15 using the axiom $hasAge(peter, 15)$; this is equivalent to $hasAge(peter, a_{15}) \wedge =_{15}(a_{15})$, where $=_{15}$ is a concrete domain predicate with the interpretation $\{15\}$.

Extending first-order logic with a concrete domain is significantly simplified if the interpretation of concrete objects is separated from the interpretation of other objects. Hence, we assume that the set of sorts $\mathcal{S}$ of a signature $\Sigma$ contains the sort $\mathsf{c}$ for the *concrete objects*, which is different from other *abstract objects*. When there is a need to distinguish the sorts syntactically, we denote the variables (general function symbols) of sort $\mathsf{c}$ with $x^{\mathsf{c}}$ ($f^{\mathsf{c}}$).

**Definition 3.2.2.** *A signature* $\Sigma = (\mathcal{P}, \mathcal{F}, \mathcal{V}, \mathcal{S})$ *is* compatible *with an admissible concrete domain* $\mathbf{D}$ *if it satisfies the following conditions:*

- $\mathsf{c} \in \mathcal{S}$ *is a* concrete *sort;*
- $\Phi_{\mathbf{D}} \subseteq \mathcal{P}$*;*
- *The signature of each predicate from* $\Phi_{\mathbf{D}}$ *is* $\mathsf{c} \times \ldots \times \mathsf{c}$*;*
- *For each general function symbol* $f$ *with a signature* $r_1 \times \ldots \times r_n \to r$*, we have* $r_i \neq \mathsf{c}$ *for each* $i$*.*

*We assume that the concrete predicate* $\approx_{\mathbf{D}}$ *is used in formulae instead of* $\approx_{\mathsf{c}}$ *for expressing equality between concrete terms. An interpretation* $I$ *over the* $\mathbf{D}$*-signature* $\Sigma$ *is a* $\mathbf{D}$-interpretation *if* $\mathcal{D}_{\mathsf{c}} = \triangle_{\mathbf{D}}$*, and, for each concrete predicate* $d \in \Phi_{\mathbf{D}}$*,* $d^I = d^{\mathbf{D}}$*. The usual notions of models, validity, satisfiability, and entailment are generalized to* $\mathbf{D}$*-models,* $\mathbf{D}$*-validity,* $\mathbf{D}$*-satisfiability, and* $\mathbf{D}$*-entailment (written* $\models_{\mathbf{D}}$*) as usual.*

When ambiguity does not arise, we do not stress $\mathbf{D}$ for satisfiability, equisatisfiability, entailment, and so on. Next, we discuss the restrictions of Definition 3.2.2.

First, the concrete domain does not specify the interpretation for general function symbols of sort $\mathsf{c}$. Hence, to simplify the treatment, we prohibit nesting of concrete terms into other (concrete or nonconcrete) terms. Hence, for a term $t$, we have $\mathsf{sort}(t|_p) = \mathsf{c}$ only if $p = \epsilon$.

Second, the equality and inequality between concrete terms are expressed only using the concrete predicates $\approx_{\mathbf{D}}$ and $\not\approx_{\mathbf{D}}$, respectively. Note that both $\approx_{\mathbf{D}}$ and $\not\approx_{\mathbf{D}}$ are concrete predicates without any special status; also, a literal $s^{\mathsf{c}} \not\approx_{\mathbf{D}} t^{\mathsf{c}}$ is a positive literal with a concrete predicate $\not\approx_{\mathbf{D}}$. If a literal $s^{\mathsf{c}} \circ t^{\mathsf{c}}$ with $\circ \in \{\approx_{\mathbf{D}}, \not\approx_{\mathbf{D}}\}$ is to be represented as an $\mathcal{E}$-term (for example, in order to apply basic superposition), it is encoded as a positive literal $(s^{\mathsf{c}} \circ t^{\mathsf{c}}) \approx \mathsf{T}$. Therefore, $\approx_{\mathbf{D}}$ and $\not\approx_{\mathbf{D}}$ are opaque to equational calculi, such as basic superposition, and they are handled only by the extensions we present in Subsection 6.1.1.

Third, practical applications require several different concrete domains at once. An approach for combining two or more concrete domains into one concrete domain has been presented in [5]. Therefore, without loss of generality we can consider only one concrete domain at the time.

### 3.2.2 The Description Logic $\mathcal{SHIQ}(\mathbf{D})$

We now present the formal definition of the description logic $\mathcal{SHIQ}(\mathbf{D})$, which is obtained by combining $\mathcal{SHIQ}$ with concrete datatypes. The syntax of $\mathcal{SHIQ}$ from Definition 3.1.1 is extended as follows:

**Definition 3.2.3.** *Let $N_{R_c}$ be the set of* concrete roles. *Additionally to $\mathcal{SHIQ}$ RBox axioms, a $\mathcal{SHIQ}(\mathbf{D})$ RBox $KB_{\mathcal{R}}$ can contain a finite number concrete role inclusion axioms $T \sqsubseteq U$.*[1]

*Let $\mathbf{D}$ be an admissible concrete domain. In addition to $\mathcal{SHIQ}$ concepts, the set of $\mathcal{SHIQ}(\mathbf{D})$ concepts contains $\exists T_1, \ldots, T_m.d$, $\forall T_1, \ldots, T_m.d$, $\leq n\, T$, and $\geq n\, T$, for $T_{(i)}$ concrete roles, $d$ an $m$-ary concrete domain predicate, and $n$ an integer. A $\mathcal{SHIQ}(\mathbf{D})$ TBox $KB_{\mathcal{T}}$ is defined analogously to Definition 3.1.1.*

*Let $N_{I_c}$ be a set of* concrete individuals. *Additionally to $\mathcal{SHIQ}$ ABox axioms, a $\mathcal{SHIQ}(\mathbf{D})$ ABox $KB_{\mathcal{A}}$ can contain a finite number of concrete role membership axioms $(\neg)T(a, b^{\mathsf{c}})$, and* (in)equality axioms *$a^{\mathsf{c}} \approx b^{\mathsf{c}}$ and $a^{\mathsf{c}} \not\approx b^{\mathsf{c}}$, where $T$ is a concrete role, $a$ is an abstract individual, and $a^{\mathsf{c}}$ and $b^{\mathsf{c}}$ are concrete individuals.*

*A $\mathcal{SHIQ}(\mathbf{D})$ knowledge base is defined as in Definition 3.1.1, where the sets $N_{R_a}$, $N_{R_c}$, $N_C$, $N_{I_a}$, and $N_{I_c}$ are mutually disjoint.*

The semantics of $\mathcal{SHIQ}$ from Definition 3.1.2 is extended to $\mathcal{SHIQ}(\mathbf{D})$ as follows:

**Definition 3.2.4.** *The semantics of a $\mathcal{SHIQ}(\mathbf{D})$ knowledge base $KB$ is given by extending the mapping $\pi$ from Definition 3.1.2 to translate $KB$ into a multi-sorted first-order formula, as shown in Table 3.4. We assume the existence of a separate sort* $\mathsf{a}$ *for abstract objects, which is different from* $\mathsf{c}$. *Atomic concept predicates have the signature* $\mathsf{a}$, *abstract roles have the signature* $\mathsf{a} \times \mathsf{a}$, *concrete roles have the signature* $\mathsf{a} \times \mathsf{c}$, *and $n$-ary concrete domain predicates have the signature* $\mathsf{c} \times \ldots \times \mathsf{c}$.

*The basic inference problem for $\mathcal{SHIQ}(\mathbf{D})$ is checking* satisfiability *of $KB$—that is, determining whether a $\mathbf{D}$-model of $\pi(KB)$ exists. The other inference problems are defined analogously to Definition 3.1.2.*

The direct model-theoretic semantics is easily extended to $\mathcal{SHIQ}(\mathbf{D})$, by extending the interpretation function $\cdot^I$ to assign an element $(a^{\mathsf{c}})^I \in \triangle_{\mathbf{D}}$ to each concrete individual $a^{\mathsf{c}}$, and a relation $T^I \subseteq \triangle^I \times \triangle_{\mathbf{D}}$ to each concrete role $T$. The semantics of new concepts and axioms is given in Table 3.5, where $T_{(i)}$ and $U$ are concrete roles and $d$ is a concrete predicate.

$\mathcal{SHIQ}(\mathbf{D})$ concepts can be rewritten into negation-normal form as follows:

$$\begin{array}{rcl@{\qquad}rcl} \neg(\exists T_1, \ldots, T_n.d) & \rightsquigarrow & \forall T_1, \ldots, T_n.\overline{d} & \neg(\forall T_1, \ldots, T_n.d) & \rightsquigarrow & \exists T_1, \ldots, T_n.\overline{d} \\ \neg(\geq (n+1)\, T) & \rightsquigarrow & \leq n\, T & \neg(\leq n\, T) & \rightsquigarrow & \geq (n+1)\, T \\ \neg(\geq 0\, T) & \rightsquigarrow & \bot & & & \end{array}$$

The size of a $\mathcal{SHIQ}(\mathbf{D})$ knowledge base $KB$ is obtained by extending the definition from Section 3.1 to handle the new constructs in the following way:

$$\begin{array}{rcl@{\qquad}rcl} |\exists T_1, \ldots, T_m.d| & = & 2 + m & |\geq n\, T| & = & n + 2 \\ |\forall T_1, \ldots, T_m.d| & = & 2 + m & |\leq n\, T| & = & n + 2 \end{array}$$

The notion of positions is extended to the new $\mathcal{SHIQ}(\mathbf{D})$ constructs as follows:

[1] Inverse concrete roles do not make sense semantically, so we do not distinguish between concrete roles and concrete role names. Furthermore, transitive concrete roles also do not make sense semantically, so they are not allowed. Note that this makes all concrete roles simple.

Table 3.4: Semantics of $\mathcal{SHIQ}(\mathbf{D})$ by Mapping to FOL

| | | |
|---|---|---|
| Mapping Concepts to FOL | | |
| $\pi_y(\forall T_1, \ldots, T_m.d, X)$ | $=$ | $\forall y_1^{\mathsf{c}}, \ldots, y_m^{\mathsf{c}} : \bigwedge_{i=1}^{m} T_i(X, y_i^{\mathsf{c}}) \rightarrow d(y_1^{\mathsf{c}}, \ldots, y_m^{\mathsf{c}})$ |
| $\pi_y(\exists T_1, \ldots, T_m.d, X)$ | $=$ | $\exists y_1^{\mathsf{c}}, \ldots, y_m^{\mathsf{c}} : \bigwedge_{i=1}^{m} T_i(X, y_i^{\mathsf{c}}) \wedge d(y_1^{\mathsf{c}}, \ldots, y_m^{\mathsf{c}})$ |
| $\pi_y(\leq n\, T, X)$ | $=$ | $\forall y_1^{\mathsf{c}}, \ldots, y_{n+1}^{\mathsf{c}} : \bigwedge_{i=1}^{n+1} T(X, y_i^{\mathsf{c}}) \rightarrow \bigvee_{i=1}^{n+1} \bigvee_{j=i+1}^{n+1} y_i^{\mathsf{c}} \approx_{\mathbf{D}} y_j^{\mathsf{c}}$ |
| $\pi_y(\geq n\, T, X)$ | $=$ | $\exists y_1^{\mathsf{c}}, \ldots, y_n^{\mathsf{c}} : \bigwedge_{i=1}^{n} T(X, y_i^{\mathsf{c}}) \wedge \bigwedge_{i=1}^{n} \bigwedge_{j=i+1}^{n} y_i^{\mathsf{c}} \not\approx_{\mathbf{D}} y_j^{\mathsf{c}}$ |
| Mapping Axioms to FOL | | |
| $\pi(T \sqsubseteq U)$ | $=$ | $\forall x, y^{\mathsf{c}} : T(x, y^{\mathsf{c}}) \rightarrow U(x, y^{\mathsf{c}})$ |
| $\pi(T(a, b^{\mathsf{c}}))$ | $=$ | $T(a, b^{\mathsf{c}})$ |
| $\pi(\neg T(a, b^{\mathsf{c}}))$ | $=$ | $\neg T(a, b^{\mathsf{c}})$ |
| $\pi(a^{\mathsf{c}} \circ b^{\mathsf{c}})$ | $=$ | $a^{\mathsf{c}} \circ_{\mathbf{D}} b^{\mathsf{c}}$ for $\circ \in \{\approx, \not\approx\}$ |

- For $C = \bowtie T_1, \ldots, T_m.d$ with $\bowtie \in \{\exists, \forall\}$, $C|_i = T_i$ for $1 \leq i \leq m$, and $C|_{m+1} = d$;
- For $C = \bowtie n\, T$ with $\bowtie \in \{\leq, \geq\}$, $C|_1 = n$ and $C|_2 = T$.

Since the new constructs do not contain nested concepts, the notion of polarity carries over from $\mathcal{SHIQ}$ to $\mathcal{SHIQ}(\mathbf{D})$ without change.

### 3.2.3 Example

To demonstrate how concrete domains can be applied in practice, we extend the configuration example from Subsection 3.1.1 with a concrete domain $\mathbf{N}$, where the domain is the set of all nonnegative integers, and the predicates have the form $\geq_{100}$ and are interpreted in the obvious way.

An important aspect of any item catalog is the item's price, which we represent using the role *price*. In a more realistic scenario, one would probably represent a price with the pair specifying the amount and the currency; however, for simplicity, we assume that all prices are represented in euros. An item has at most one price, so we declare *price* to be functional by (3.18).

$$\top \sqsubseteq \leq 1\, \mathit{price} \tag{3.18}$$

We now represent additional classes of items in our catalog. For example, using (3.19) we state that high-end PCs are either graphics workstations, or PCs with a price higher than 2000 EUR.

$$\mathit{HighEnd} \sqsubseteq \mathit{GrWS} \sqcup (\mathit{PC} \sqcap \exists \mathit{price}.{\geq_{2000}}) \tag{3.19}$$

We denote with $KB_3$ the knowledge base $KB_1$ from Subsection 3.1.1 extended with axioms (3.18) and (3.19).

Table 3.5: Direct Model-Theoretic Semantics of $\mathcal{SHIQ}(\mathbf{D})$

| Interpreting Concepts | | |
|---|---|---|
| $(\forall T_1, \ldots, T_m.d)^I$ | $=$ | $\{x \mid \forall y_1, \ldots, y_m : (x, y_1) \in T_1^I \wedge \ldots \wedge (x, y_m) \in T_m^I \rightarrow (y_1, \ldots, y_m) \in d^{\mathbf{D}}\}$ |
| $(\exists T_1, \ldots, T_m.d)^I$ | $=$ | $\{x \mid \exists y_1, \ldots, y_m : (x, y_1) \in T_1^I \wedge \ldots \wedge (x, y_m) \in T_m^I \wedge (y_1, \ldots, y_m) \in d^{\mathbf{D}}\}$ |
| $(\leq n\, T)^I$ | $=$ | $\{x \mid \sharp\{y \mid (x, y) \in T^I\} \leq n\}$ |
| $(\geq n\, T)^I$ | $=$ | $\{x \mid \sharp\{y \mid (x, y) \in T^I\} \geq n\}$ |
| **Semantics of Axioms** | | |
| $T \sqsubseteq U$ | | $T^I \subseteq U^I$ |
| $T(a, b^{\mathsf{c}})$ | | $(a^I, (b^{\mathsf{c}})^I) \in T^I$ |
| $\neg T(a, b^{\mathsf{c}})$ | | $(a^I, (b^{\mathsf{c}})^I) \notin T^I$ |
| $a^{\mathsf{c}} \approx b^{\mathsf{c}}$ | | $(a^{\mathsf{c}})^I = (b^{\mathsf{c}})^I$ |
| $a^{\mathsf{c}} \not\approx b^{\mathsf{c}}$ | | $(a^{\mathsf{c}})^I \neq (b^{\mathsf{c}})^I$ |

Let us now assume that a customer wants to buy a high-end PC with a price of 1500 EUR at most. These choices are represented by axioms (3.20) and (3.21); let $KB_4$ be the knowledge base containing them.

$$HighEnd(pc_2) \tag{3.20}$$

$$\exists price.{\leq_{1500}}(pc_2) \tag{3.21}$$

Based on the user's choices, the configuration system can conclude that $pc_2$ must be a graphics workstation:

$$KB_3 \cup KB_4 \models GrWS(pc_2) \tag{3.22}$$

Namely, $pc_2$ is declared to be a high-end PC with a price lower than 1500 EUR. Since each PC has only one price, the price cannot be also higher than 2000 EUR; formally stated, the conjunction of concrete domain predicates $\geq_{2000}(x) \wedge \leq_{1500}(x)$ is unsatisfiable. Hence, if axioms (3.19) and (3.19) are to be satisfied, $pc_2$ must be a graphics workstation.

# Part II

# From Description Logics to Disjunctive Datalog

# Chapter 4

# Reduction Algorithm at a Glance

The algorithm for reducing a $\mathcal{SHIQ}(\mathbf{D})$ knowledge base to a disjunctive datalog program is fairly complex and technical. This is so because $\mathcal{SHIQ}(\mathbf{D})$ encompasses different features, each of which raises special concerns for resolution-based theorem proving. For example, handling number restrictions requires equality reasoning, which in turn requires a complex calculus, such as basic superposition. Similarly, a new approach for reasoning with concrete domains is required to handle datatypes.

To make our work easier to follow, in this chapter we present a simpler version of all algorithms, scaled down to the basic description logic $\mathcal{ALC}$. Even though it is quite simple, $\mathcal{ALC}$ is an ExpTime-complete logic [144], with features common to most of the more complex DLs, such as existential and universal quantification, and general concept inclusion axioms. The reduction algorithm for $\mathcal{ALC}$ conveys all the basic ideas, without overloading the presentation with details. We hope that this will help the reader to understand the intuition behind the algorithms presented in the following chapters.

This chapter is of tutorial character, and as such we tried to make it self-contained (apart from the general definitions that we presented in Chapter 2). However, it is not strictly necessary for understanding the other chapters and may be skipped. To make other chapters self-contained as well, some definitions and clarifications are repeated in the latter chapters.

## 4.1 The Main Difficulty in Reducing DLs to Datalog

For an $\mathcal{ALC}$ knowledge base $KB$, our goal is to derive a disjunctive datalog program $\mathsf{DD}(KB)$ such that $KB \models \alpha$ if and only if $\mathsf{DD}(KB) \models \alpha$ for $\alpha$ of the form $A(a)$ or $R(a, b)$. In other words, $KB$ and $\mathsf{DD}(KB)$ should entail the same set of positive ground facts. We can thus use $\mathsf{DD}(KB)$ instead of $KB$ for query answering, and in doing so we can apply all optimization techniques known in deductive databases.

As shown by Definition 3.1.2 and [21], there is a close correspondence between description logics and first-order logic. Consider the following knowledge base:

$$KB = \{A \sqsubseteq \exists R.A, \exists R.\exists R.A \sqsubseteq B, A(a)\} \tag{4.1}$$

A naïve attempt to reduce $KB$ into disjunctive datalog is to translate $KB$ into first-order logic as $\pi(KB)$, skolemize it, translate it into conjunctive normal form, and rewrite the obtained set of clauses into rules. On $KB$, such an approach produces the following logic program $\mathsf{LP}(KB)$:

$$R(x, f(x)) \leftarrow A(x) \tag{4.2}$$

$$A(f(x)) \leftarrow A(x) \tag{4.3}$$

$$B(x) \leftarrow R(x, y), R(y, z), A(z) \tag{4.4}$$

$$A(a) \tag{4.5}$$

Clearly, $KB$ and $\mathsf{LP}(KB)$ entail the same set of ground facts. However, $\mathsf{LP}(KB)$ contains a function symbol in a recursive rule (4.3). This raises the issue of how to answer queries in $\mathsf{LP}(KB)$. Namely, well-known query evaluation techniques, such as bottom-up saturation, will not terminate on $\mathsf{LP}(KB)$: using bottom-up saturation, we shall derive $A(f(a))$, $R(a, f(a))$, $A(f(f(a)))$, $R(f(a), f(f(a)))$, $B(a)$, and so on. Obviously, such an algorithm will continue deriving ever deeper facts, and will therefore never terminate. Note that we need all previously derived facts to derive $B(a)$ from $\mathsf{LP}(KB)$, and that we do not know a priori when all relevant ground facts have been derived, so that we might stop the saturation.

This problem could be solved by employing an appropriate cycle detection mechanism. The authors use such an approach in [49] to derive a hyperresolution decision procedure for a variant of the guarded fragment. However, using specialized algorithms for evaluating queries in $\mathsf{LP}(KB)$ takes us further away from our original goal of applying deductive database optimization techniques to description logics. In a way, such an algorithm could be understood as an alternative syntactic notation for the tableau calculus, for which it is still unclear how to apply known optimization techniques, such as magic sets.

To avoid potential problems with termination, our goal is to derive a true disjunctive datalog program $\mathsf{DD}(KB)$ without function symbols. For such a program, queries can be evaluated using any standard technique; furthermore, all existing optimization techniques known in deductive databases can be applied directly. Hence, the main problem that we deal with is the elimination of function symbols from $\mathsf{LP}(KB)$.

## 4.2 The General Idea

To obtain the reduction of an $\mathcal{ALC}$ knowledge base $KB$ to a disjunctive datalog program entailing the same set of ground facts, we start off with a slightly simpler task of deriving a program $\mathsf{DD}(KB)$ that is satisfiable if and only if $KB$ is satisfiable. The

intuitive principle used to ensure the equisatisfiability of $KB$ and $\mathsf{DD}(KB)$ is relatively simple. Let us assume that unsatisfiability of $KB$ can be demonstrated by a refutation in some sound and complete calculus $\mathcal{C}$. If it is possible to simulate inferences of $\mathcal{C}$ on $KB$ using a datalog program $\mathsf{DD}(KB)$, a refutation in $KB$ by $\mathcal{C}$ can be reduced to a refutation in $\mathsf{DD}(KB)$. Conversely, if $\mathsf{DD}(KB)$ is unsatisfiable, there is a refutation in $\mathsf{DD}(KB)$. If it is possible to simulate inferences in $\mathsf{DD}(KB)$ by the calculus $\mathcal{C}$ on $KB$, then a refutation in $\mathsf{DD}(KB)$ can be reduced to a refutation in $KB$. To summarize, if inferences can be simulated in both directions, $\mathsf{DD}(KB)$ and $KB$ are equisatisfiable.

To obtain a sound, complete, and terminating algorithm from the high-level idea outlined in the previous paragraph, it is necessary to select an appropriate calculus $\mathcal{C}$, capable of effectively deciding satisfiability of $KB$. Positive disjunctive datalog is strongly related to clausal first-order logic. Intuitively, simulating inferences in disjunctive datalog is easier if $\mathcal{C}$ is a clausal refutation calculus. Therefore, we first derive a decision procedure for $\mathcal{ALC}$ based on the ordered resolution calculus $\mathcal{R}$ (for a definition of $\mathcal{R}$, see Section 2.4). In particular, we show in Section 4.3 how to translate $KB$ into an equisatisfiable set of clauses $\Xi(KB)$, and in Section 4.4 that exhaustive application of the inference rules of $\mathcal{R}$ on $\Xi(KB)$ eventually terminates. Since $\mathcal{R}$ is sound and complete, termination implies that $\mathcal{R}$ decides satisfiability of $\Xi(KB)$, and therefore of $KB$ as well.

Based on such a procedure, we derive in Section 4.5 the desired reduction of $KB$ to a disjunctive datalog program. It turns out that, for $\mathcal{ALC}$, simulating inferences of $\mathcal{R}$ in disjunctive datalog and vice versa is quite straightforward.

After presenting several simple examples in Section 4.6, in Section 4.7 we summarize the issues that need to be dealt with in order to extend the algorithms to $\mathcal{SHIQ}(\mathbf{D})$. Finally, in Section 4.8 we discuss some interesting aspects of our algorithms.

## 4.3 Translating $KB$ into Clauses

The first step in deciding satisfiability of an $\mathcal{ALC}$ knowledge base $KB$ by $\mathcal{R}$ is to transform $KB$ into an equisatisfiable set of clauses $\Xi(KB)$. A straightforward way of doing so is to compute the formula $\pi(KB)$ (where $\pi$ is the operator from Definition 3.1.2), and then to apply the clausification operator $\mathsf{Cls}$ to it (that is, to skolemize $\pi(KB)$ and to transform the result into conjunctive normal form using well-known identities, as explained in Section 2.1).

However, such an algorithm has two important drawbacks. First, the size of $\mathsf{Cls}(\pi(KB))$ could be exponential in the size of $\pi(KB)$, due to nesting of $\sqcap$ and $\sqcup$ connectives. Second, since $KB$ is an $\mathcal{ALC}$ knowledge base, the formula $\pi(KB)$ is of a particular syntactic structure, which we exploit to derive the decision procedure. Direct clausification of $\pi(KB)$ would destroy this structure, and thus make it difficult, if not impossible, to obtain a decision procedure for $\mathcal{ALC}$ by $\mathcal{R}$.

In order to avoid the exponential blowup and to preserve the structure of formulae, we apply the *structural transformation*, introduced in [108, 99, 8]. Next, we present an alternative, but equivalent definition.

**Definition 4.3.1.** *Let $C$ be an $\mathcal{ALC}$ concept in negation-normal form. A position $p \neq \epsilon$ in $C$ is* eligible for replacement *if $C|_p$ is of form $\bigsqcup L_i$, $\bigsqcap L_i$, $\forall R.L$, or $\exists R.L$, where $L_{(i)}$ are all literal concepts. The* definitorial form *of $C$, written $\mathsf{Def}(C)$, is defined recursively as follows:*

$$\mathsf{Def}(C) = \begin{cases} \{C\} & \text{if } C \text{ is a literal concept} \\ \{\neg Q \sqcup C|_p\} \cup \mathsf{Def}(C[Q]_p) & \text{if } p \text{ is eligible for replacement in } C \end{cases}$$

For example, $\mathsf{Def}(\exists R.(\forall S.\neg A)) = \{\exists R.Q,\ \neg Q \sqcup \forall S.\neg A\}$. Note that $\neg Q \sqcup \forall S.\neg A$ can be interpreted as $Q \sqsubseteq \forall S.\neg A$, which allows us to use $Q$ as a new name for $\forall S.\neg A$ in $\exists R.(\forall S.\neg A)$. Furthermore, as shown by the following lemma, this transformation does not affect satisfiability: $\top \sqsubseteq \exists R.(\forall S.\neg A)$ and $\{\top \sqsubseteq \exists R.Q, \top \sqsubseteq \neg Q \sqcup \forall S.\neg A\}$ are equisatisfiable.

**Lemma 4.3.2.** *For an $\mathcal{ALC}$ concept $C$ in negation-normal form, the axiom $\top \sqsubseteq C$ is satisfiable if and only if the set of axioms $\{\top \sqsubseteq D_i \mid D_i \in \mathsf{Def}(C)\}$ is satisfiable.*

*Proof.* The proof is by induction on the number of recursive invocations of $\mathsf{Def}$. The induction base is trivial, so we consider the induction step, which replaces the subconcept $C|_p$ in $C$ with $Q$, resulting in $D = C[Q]_p$. For the ($\Rightarrow$) direction, let $I$ be a model satisfying $\top \sqsubseteq C$. Obviously, an interpretation $I'$, obtained by extending $I$ with $Q^{I'} = D^I$, satisfies $\top \sqsubseteq D$ and $\top \sqsubseteq \neg Q \sqcup C|_p$. For the ($\Leftarrow$) direction, let $I$ be a model of $\top \sqsubseteq D$ and $\top \sqsubseteq \neg Q \sqcup C|_p$. The following property can be shown by an easy induction on the structure of $D$: for each position $q$ in $D$, $(D|_q)^I \subseteq (C|_q)^I$. But then, for $q = \epsilon$, we have $D^I \subseteq C^I$. Since $\triangle^I \subseteq D^I$, we have that $I$ is a model of $\top \sqsubseteq C$. □

The set of clauses $\Xi(KB)$, encoding an $\mathcal{ALC}$ knowledge base $KB$ in first-order logic, is defined as follows:

**Definition 4.3.3.** *We extend the clausification operator $\mathsf{Cls}$ from Section 2.1 to $\mathcal{ALC}$ concepts as follows:*

$$\mathsf{Cls}(C) = \bigcup_{D \in \mathsf{Def}(\mathsf{NNF}(C))} \mathsf{Cls}(\forall x : \pi_y(D, x))$$

*For an extensionally reduced $\mathcal{ALC}$ knowledge base $KB$, $\Xi(KB)$ is the smallest set of clauses satisfying the following conditions:*

- *For each ABox axiom $\alpha$ in $KB$, $\mathsf{Cls}(\pi(\alpha)) \subseteq \Xi(KB)$;*
- *For each TBox axiom $C \sqsubseteq D$ in $KB$, $\mathsf{Cls}(\neg C \sqcup D) \subseteq \Xi(KB)$;*
- *For each TBox axiom $C \equiv D$ in $KB$, $\mathsf{Cls}(\neg C \sqcup D) \subseteq \Xi(KB)$ and $\mathsf{Cls}(\neg D \sqcup C) \subseteq \Xi(KB)$.*

*If $KB$ is not extensionally reduced, then $\Xi(KB) = \Xi(KB')$, where $KB'$ is an extensionally reduced knowledge base obtained from $KB$ as explained in Section 3.1.*

Table 4.1: Clause Types after Preprocessing

| | |
|---|---|
| 1 | $\bigvee(\neg)C_i(x) \vee R(x, f(x))$ |
| 2 | $\bigvee(\neg)C_i(x) \vee (\neg)D(f(x))$ |
| 3 | $\bigvee(\neg)C_i(x)$ |
| 4 | $\bigvee(\neg)C(x) \vee \neg R(x, y) \vee (\neg)D(y)$ |
| 5 | $(\neg)C(a)$ |
| 6 | $(\neg)R(a, b)$ |

We now show that clausification does not affect the semantics of a knowledge base, and that it produces clauses of a certain syntactic structure:

**Lemma 4.3.4.** *Let $KB$ be an $\mathcal{ALC}$ knowledge base. Then, the following claims hold:*

- *$KB$ is satisfiable if and only if $\Xi(KB)$ is satisfiable.*
- *$\Xi(KB)$ can be computed in time polynomial in $|KB|$.*
- *Each clause in $\Xi(KB)$ is of one of the types given in Table 4.1.*

*Proof.* Equisatisfiability of $KB$ and $\Xi(KB)$ is an easy consequence of Lemma 4.3.2. Furthermore, the number of recursive invocations of $\mathsf{Def}$ and the number of new concepts $Q$ are linear in the number of subconcepts of $C$. Hence, $|\mathsf{Def}(C)|$ is linear in $|C|$, so $|\Xi(KB)|$ is polynomial in $|KB|$. Finally, observe that $\mathsf{Def}(C)$ can contain only concepts of the form $D$ or $\neg Q \sqcup D$, where $D$ is of the form $\bigsqcup L_i$, $\bigsqcap L_i$, $\forall R.L$, or $\exists R.L$, and all $L_{(i)}$ are literal concepts. Hence, for $D = \exists R.L$, $\mathsf{Cls}(D)$ contains clauses of type 1 and 2; for $D = \forall R.L$ a clause of type 4; and for $D = \bigsqcup L_i$ or $D = \bigsqcap L_i$ clauses of type 3. Clauses of type 5 and 6 are obtained by clausifying ABox axioms. □

## 4.4 Deciding Satisfiability of $\Xi(KB)$ by $\mathcal{R}$

Since $\mathcal{R}$ is a sound and complete calculus, we can use it to check satisfiability of $\Xi(KB)$. To obtain a decision procedure, we just need to ensure that each saturation of $\Xi(KB)$ by $\mathcal{R}$ terminates. Joyner outlined the basic principle to achieve this in [80]: we simply ensure that the number of clauses that can be derived in a saturation is bounded.

There are two main reasons why deriving an infinite number of clauses from a set of clauses $N$ might be possible. First, we might keep deriving clauses with deeper and deeper terms. Consider $N_1 = \{C(a), \neg C(x) \vee C(f(x))\}$. If we select $\neg C(x)$ in the second clause and apply resolution, we derive $C(f(a))$, $C(f(f(a)))$, and so forth. Second, we might keep deriving clauses containing more and more variables. Consider $N_2 = \{\neg R(x, y) \vee \neg R(y, z) \vee R(x, z), C(x) \vee R(x, y) \vee D(y), E(x) \vee R(x, y) \vee F(y)\}$. If we select $\neg R(x, y)$ and $\neg R(y, z)$ in the first clause and apply resolution twice, we

obtain $C(x) \vee D(y) \vee E(y) \vee F(z) \vee R(x, z)$, which contains more variables than the side premises involved in the inference. It is easy to see that further inferences with this clause will additionally increase the number of variables.

The resolution inferences performed on certain clauses are determined by the parameters of the calculus—namely, the literal ordering and the selection function. Hence, by choosing these parameters appropriately, we can often restrict the resolution inferences in a way that allows us to establish a bound on the term depth and on the number of variables. Consider again the set of clauses $N_1$: instead of selecting $\neg C(x)$, if we ensure that $C(f(x)) \succ \neg C(x)$, then the second clause can participate in an inference only on literal $C(f(x))$. Furthermore, $C(f(x))$ and $C(a)$ do not unify, so no inference of $\mathcal{R}$ is applicable to $N_1$. Hence, $N_1$ is saturated, and, since it does not contain the empty clause, it is satisfiable. In the following definition, we choose the parameters for $\mathcal{R}$ that achieve such an effect on clauses from $\Xi(KB)$.

**Definition 4.4.1.** *Let $\mathcal{R}_{DL}$ denote the calculus $\mathcal{R}$ parameterized as follows:*

- *The literal ordering is an admissible ordering $\succ$ such that $R(x, f(x)) \succ \neg C(x)$ and $D(f(x)) \succ \neg C(x)$, for all function symbols $f$, and predicates $R$, $C$, and $D$.*
- *The selection function selects every negative binary literal in each clause.*

An ordering satisfying Definition 4.4.1 can be obtained by instantiating an appropriate lexicographic path ordering. Technically speaking, negative literals are not terms, but for the purposes of comparing literals by an LPO, we consider $\neg$ to be a unary general function symbol. To make an LPO compatible with $\mathcal{R}_{DL}$, we choose any precedence $>$ over predicate and general function symbols such that $f > P > \neg$, for each function symbol $f$ and each predicate symbol $P$. Namely, LPOs have the subterm property, so $\neg A \succ A$; furthermore, since $\neg$ is the smallest symbol in $>$, we cannot have $\neg A \succ B \succ A$. Hence, such an LPO is admissible for $\mathcal{R}$. Moreover, by the definition of LPOs from Section 2.2, it is easy to check that $D(f(x)) \succ f(x) \succ \neg C(x)$, and $R(x, f(x)) \succ f(x) \succ \neg C(x)$.

To obtain a decision procedure for $\mathcal{ALC}$, we next identify the type of clauses that can be derived in a saturation of $\Xi(KB)$ by $\mathcal{R}_{DL}$. We generalize the clauses from Table 4.1 to *$\mathcal{ALC}$-clauses*, presented in Table 4.2. For a term $t$, with $\mathbf{P}(t)$ we denote a possibly empty disjunction of the form $(\neg)P_1(t) \vee \ldots \vee (\neg)P_n(t)$. With $\mathbf{P}(\mathbf{f}(x))$ we denote a possibly empty disjunction of the form $\mathbf{P_1}(f_1(x)) \vee \ldots \vee \mathbf{P_m}(f_m(x))$. Note that this definition allows each $\mathbf{P_i}(f_i(x))$ to contain positive and negative literals.

We now prove the following central lemma:

**Lemma 4.4.2.** *Each $\mathcal{R}_{DL}$ inference, when applied to $\mathcal{ALC}$-clauses, produces an $\mathcal{ALC}$-clause.*

*Proof.* The lemma can easily be proved by considering all possible $\mathcal{R}_{DL}$ inferences on all types of $\mathcal{ALC}$-clauses, which are summarized in Table 4.3. For the sake of brevity, we omit inferences in which participating literals are complemented. The notation

Table 4.2: Types of $\mathcal{ALC}$-Clauses

| | |
|---|---|
| 1 | $\mathbf{P}(x) \vee R(x, f(x))$ |
| 2 | $\mathbf{P_1}(x) \vee \mathbf{P_2}(\mathbf{f}(x))$ |
| 3 | $\mathbf{P_1}(x) \vee \neg R(x, y) \vee \mathbf{P_2}(y)$ |
| 4 | $\mathbf{P}(\mathbf{a})$ |
| 5 | $(\neg)R(a, b)$ |

$(n + m = k)$ next to each inference specifies that the inference premises are of types $n$ and $m$, and the conclusion is of type $k$. Observe that, due to the requirement on the literal ordering $\succ$, a literal of the form $(\neg)A(x)$ occurring in a clause $C$ can participate in an inference only if $C$ does not contain a literal of the form $(\neg)B(f(x))$ or $R(x, f(x))$. Furthermore, a ground literal $A(a)$ does not unify with a literal $A(f(x))$, and $R(a, b)$ does not unify with $R(x, f(x))$. Hence, ground clauses can participate only in inferences with clauses not containing terms of the form $f(x)$. □

As shown by the following lemma, for a finite knowledge base $KB$, the number of $\mathcal{ALC}$-clauses is finite. In fact, the bound on the number of derivable clauses can be used to estimate the complexity of the algorithm.

**Lemma 4.4.3.** *For an $\mathcal{ALC}$ knowledge base $KB$, the longest $\mathcal{ALC}$-clause over the signature of $\Xi(KB)$ is polynomial in $|KB|$, and the number of such clauses different up to variable renaming is exponential in $|KB|$.*

*Proof.* Let $c$ be the number of unary predicates, $f$ the number of unary function symbols, and $i$ the number of constants in the signature of $\Xi(KB)$. Then, $c$ is linear in $|KB|$, since each concept introduced in $\mathsf{Def}(C)$ corresponds to one nonliteral subconcept of $C$. Similarly, $f$ is linear in $|KB|$, since each function symbol is introduced by skolemizing one concept of the form $\exists R.C$. Finally, $i$ is trivially linear in $|KB|$.

Consider now the maximal $\mathcal{ALC}$-clause $Cl_2$ of type 2. Such a clause contains a possibly negated literal $A(x)$ for each unary predicate $A$, and a possibly negated literal $A(f(x))$ for each pair of unary predicate and function symbols, yielding at most $\ell = 2c + 2cf$ literals, which is polynomial in $|KB|$. Each $\mathcal{ALC}$-clause of type 2 is a subset of $Cl_2$, so there are $2^\ell$ such clauses; that is, the number of clauses is exponential in $|KB|$. For other $\mathcal{ALC}$-clause types, the bounds on the length and on the number of clauses can be derived in an analogous way. □

We now state the main result of this section:

**Theorem 4.4.4.** *For an $\mathcal{ALC}$ knowledge base $KB$, saturating $\Xi(KB)$ by $\mathcal{R}_{DL}$ decides satisfiability of $KB$ and runs in time exponential in $|KB|$.*

*Proof.* By Lemma 4.4.3, the number of clauses derivable by $\mathcal{R}_{DL}$ from $\Xi(KB)$ is exponential in $|KB|$. Each inference can be performed in time polynomial in the size of

Table 4.3: Possible Inferences by $\mathcal{R}_{DL}$ on $\mathcal{ALC}$-Clauses

$$\frac{\mathbf{P_1}(x) \vee \mathbf{P_2}(\mathbf{f}(x)) \vee \neg A(g(x)) \quad A(x) \vee \mathbf{P_3}(x)}{\mathbf{P_1}(x) \vee \mathbf{P_2}(\mathbf{f}(x)) \vee \mathbf{P_3}(g(x))} \textbf{ (2+2=2)}$$

$$\frac{\mathbf{P_1}(x) \vee \neg A(x) \quad A(x) \vee \mathbf{P_2}(x)}{\mathbf{P_1}(x) \vee \mathbf{P_2}(x)} \textbf{ (2+2=2)}$$

$$\frac{\mathbf{P_1}(x) \vee \mathbf{P_2}(\mathbf{f}(x)) \vee \neg A(g(x)) \quad A(g(x)) \vee \mathbf{P_3}(\mathbf{h}(x)) \vee \mathbf{P_4}(x)}{\mathbf{P_1}(x) \vee \mathbf{P_2}(\mathbf{f}(x)) \vee \mathbf{P_3}(\mathbf{h}(x)) \vee \mathbf{P_4}(x)} \textbf{ (2+2=2)}$$

$$\frac{\mathbf{P}(x) \vee R(x, f(x)) \quad \mathbf{P_1}(x) \vee \neg R(x, y) \vee \mathbf{P_2}(y)}{\mathbf{P}(x) \vee \mathbf{P_1}(x) \vee \mathbf{P_2}(f(x))} \textbf{ (1+3=2)}$$

$$\frac{\mathbf{P_1}(\mathbf{a}) \vee \neg A(b) \quad A(x) \vee \mathbf{P_2}(x)}{\mathbf{P_1}(\mathbf{a}) \vee \mathbf{P_2}(b)} \textbf{ (4+2=4)} \qquad \frac{\mathbf{P_1}(\mathbf{a}) \vee \neg A(b) \quad A(b) \vee \mathbf{P_2}(\mathbf{c})}{\mathbf{P_1}(\mathbf{a}) \vee \mathbf{P_2}(\mathbf{c})} \textbf{ (4+4=4)}$$

$$\frac{R(a, b) \quad \mathbf{P_1}(x) \vee \neg R(x, y) \vee \mathbf{P_2}(y)}{\mathbf{P_1}(a) \vee \mathbf{P_2}(b)} \textbf{ (5+3=4)} \qquad \frac{R(a, b) \quad \neg R(a, b)}{\square} \textbf{ (5+5=2)}$$

clauses. Subsumption checking is NP-complete in the size of clauses [52], so it can be performed in exponential time. Hence, the saturation terminates after performing at most an exponential number of steps. Since $\mathcal{R}_{DL}$ is sound and complete, it decides satisfiability of $\Xi(KB)$, and by Lemma 4.3.4 of $KB$ as well, in time that is exponential in $|KB|$. □

## 4.5 Translating $\mathcal{ALC}$ to Disjunctive Datalog

Given a decision procedure for checking satisfiability of an $\mathcal{ALC}$ knowledge base $KB$, it is now easy to obtain the desired reduction to disjunctive datalog. From Table 4.3, we see that $(i)$ a ground clause cannot participate in an inference with a nonground clause containing a function symbol, and $(ii)$ as soon as one premise in an inference by $\mathcal{R}_{DL}$ is ground, the conclusion is ground as well. Hence, we can perform all inferences among nonground clauses first, after which we can simply delete all nonground clauses containing function symbols. The remaining clause set consists of clauses without function symbols, which can easily be translated into a disjunctive datalog program,

by moving positive literals into rule heads and negative literals into rule bodies. A minor problem arises if the resulting rules contain unsafe variables—that is, variables occurring only in the head, but not in the body. We deal with such clauses by a simple trick: we introduce a new predicate $HU$ and add an assertion $HU(a)$ for each individual $a$; next, we append $HU(x)$ to the body of each rule in which $x$ is an unsafe variable. We now present this algorithm formally.

**Definition 4.5.1.** *Let $KB$ be an extensionally reduced $\mathcal{ALC}$ knowledge base. Then, $\Gamma(KB_\mathcal{T})$ is the set of clauses obtained by saturating $\Xi(KB_\mathcal{T})$ by $\mathcal{R}_{DL}$, and then deleting all clauses containing function symbols. Furthermore, the operator $\lambda$ maps clauses to clauses as follows:*

$$\lambda(C) = C \cup \{\neg HU(x) \mid \textit{ for each unsafe variable } x \textit{ in } C\ \}$$

*For a set of clauses $N$, $\lambda(N)$ is the set of clauses obtained by applying $\lambda$ to each element of $N$. The* function-free *version of $KB$ is defined as follows:*

$$\mathsf{FF}(KB) = \lambda(\Gamma(KB_\mathcal{T})) \cup \Xi(KB_\mathcal{A}) \cup \{HU(a) \mid \textit{ for each individual } a \textit{ occurring in } KB\}$$

*Finally, a disjunctive datalog program $\mathsf{DD}(KB)$ is the set of rules obtained by moving in each clause from $\mathsf{FF}(KB)$ all positive literals into the rule head, and all negative literals into the rule body.*

*If $KB$ is not extensionally reduced, then $\mathsf{DD}(KB) = \mathsf{DD}(KB')$, where $KB'$ is an extensionally reduced knowledge base obtained from $KB$ as explained in Section 3.1.*

We now state the properties of $\mathsf{DD}(KB)$:

**Theorem 4.5.2.** *Let $KB$ be an $\mathcal{ALC}$ knowledge base. Then, the following claims hold:*

1. *$KB$ is unsatisfiable if and only if $\mathsf{DD}(KB)$ is unsatisfiable.*
2. *$KB \models \alpha$ if and only if $\mathsf{DD}(KB) \models_c \alpha$, where $\alpha$ is of the form $A(a)$ or $R(a, b)$, and $A$ is an atomic concept.*
3. *$KB \models C(a)$ for a nonliteral concept $C$ if and only if, for $Q$ a new atomic concept, $\mathsf{DD}(KB \cup \{C \sqsubseteq Q\}) \models_c Q(a)$.*
4. *The number of literals in each rule in $\mathsf{DD}(KB)$ is at most polynomial, the number of rules in $\mathsf{DD}(KB)$ is at most exponential, and $\mathsf{DD}(KB)$ can be computed in time exponential in $|KB|$.*

*Proof.* (Claim 1.) Table 4.3 shows that each inference with at least one ground premise always produces a ground conclusion. Hence, in saturating $\Xi(KB)$ by $\mathcal{R}_{DL}$, it is possible to perform all inferences among nonground clauses first. Furthermore, Table 4.3 also shows that ground clauses can participate in inferences only with clauses not containing function symbols. Hence, after performing all inferences among nonground clauses of $\Xi(KB)$, one can delete all clauses containing terms of the form $f(x)$.

By Definition 4.3.3, $\Xi(KB_{\mathcal{T}})$ is exactly the set of nonground clauses of $\Xi(KB)$, so $\Gamma(KB_{\mathcal{T}})$ is exactly the set of clauses obtained by saturating the nonground part of $\Xi(KB)$ and deleting the clauses containing function symbols. Furthermore, it is easy to see that $\Gamma = \Gamma(KB_{\mathcal{T}}) \cup \Xi(KB_{\mathcal{A}})$ is satisfiable if and only if $\mathsf{FF}(KB)$ is satisfiable. Namely, both $\Gamma$ and $\mathsf{FF}(KB)$ are function-free sets of clauses, whose sets of ground instances differ only on clauses containing unsafe variables. However, for a clause $C \in \Gamma$ containing an unsafe variable $x$ and a ground substitution $\tau = \{x \mapsto a\}$, $\mathsf{FF}(KB)$ contains clauses $C \vee \neg HU(x)$ and $HU(a)$, which together imply $C\tau$. Hence, the ground instances of $\mathsf{FF}(KB)$ imply all ground instances of $\Gamma$, so, if $\mathsf{FF}(KB)$ is satisfiable, $\Gamma$ is satisfiable as well. Furthermore, since $HU$ is a new predicate not occurring in $\Gamma$, each model of $\Gamma$ can easily be extended to a model of $\mathsf{FF}(KB)$. Therefore, $\Gamma$ and $\mathsf{FF}(KB)$ are equisatisfiable.

Finally, $\mathsf{DD}(KB)$ is a positive datalog program whose rules are syntactic variants of the clauses from $\mathsf{FF}(KB)$. Hence, $\mathsf{DD}(KB)$ is satisfiable if and only if $\mathsf{FF}(KB)$ is satisfiable.

(Claim 2.) Simply observe that $KB \models \alpha$ if and only if $KB \cup \{\neg\alpha\}$ is unsatisfiable. The latter is the case if and only if $\mathsf{DD}(KB \cup \{\leftarrow \alpha\}) = \mathsf{DD}(KB) \cup \{\leftarrow \alpha\}$ is unsatisfiable, which is the case if and only if $\mathsf{DD}(KB) \models_c \alpha$.

(Claim 3.) Follows in the same manner as Claim 2.

(Claim 4.) Follows immediately from Lemma 4.4.3. □

## 4.6 Examples

We now present several rather simple examples that point out important properties of the reduction algorithm.

Readers familiar with more common approaches to DL reasoning might ask themselves how role successors are encoded in datalog programs. As we show next, role successors are not at all encoded in the datalog program. Let $KB_1 = \{C \sqsubseteq \exists R.D\}$, so $\Xi(KB_1) = \{\neg C(x) \vee R(x, f(x)), \neg C(x) \vee D(f(x))\}$. Now $\Xi(KB_1)$ is already saturated by $\mathcal{R}_{DL}$, so after removing all clauses containing function symbols, we get $\mathsf{DD}(KB_1) = \emptyset$. This may seem quite confusing: $KB_1$ implies the existence of at least one $R$-successor for each member of $C$, whereas $\mathsf{DD}(KB_1)$ does not reflect that. However, Theorem 4.5.2 is not invalidated. Namely, the individuals introduced by the existential quantifier in $KB_1$ are unnamed, so they cannot be used in queries. Hence, for an arbitrary extensionally reduced ABox $KB_{\mathcal{A}}$, $KB_1 \cup KB_{\mathcal{A}}$ does not imply any new facts of the form $A(a)$ or $R(a, b)$. Also, note that the models of $\mathsf{DD}(KB_1)$ are completely unrelated to the models of $KB_1$. Intuitively, this is so because the reduction algorithm is based on a proof-theoretic correspondence between refutations in $\Xi(KB)$ and $\mathsf{DD}(KB)$. The models of $KB$ and $\mathsf{DD}(KB)$ coincide only on positive ground facts, whereas, for unnamed individuals, the models are completely unrelated.

In order to be able to draw additional consequences from $KB_1$, we need to extend it with additional statements about $C$, $D$, or $R$. Let $KB_2 = KB_1 \cup \{D \sqsubseteq \bot\}$, so

$\Xi(KB_2) = \Xi(KB_1) \cup \{\neg D(x)\}$. Saturation of $\Xi(KB_2)$ produces one additional clause $\neg C(x)$, so we get $\mathsf{DD}(KB_2) = \{\leftarrow C(x), \leftarrow D(x)\}$. This shows that the reduction is not modular: for arbitrary knowledge bases $KB_1$ and $KB_2$, $\mathsf{DD}(KB_1) \cup \mathsf{DD}(KB_2)$ is not necessarily equal to $\mathsf{DD}(KB_1 \cup KB_2)$.

The key step in the reduction is the saturation of $\Xi(KB_\mathcal{T})$ by $\mathcal{R}_{DL}$. It computes all relevant nonground consequences of $\Xi(KB_\mathcal{T})$, which ensures that subsequent removal of clauses containing function symbols does not change the set of drawn ground consequences. One can think of $\mathcal{R}_{DL}$ as producing shortcut clauses that derive facts about objects without explicitly expanding the successors of each object. This is demonstrated by the following knowledge base:

$$KB_3 = \{A \sqsubseteq \exists R.B,\ B \sqsubseteq C,\ \exists R.C \sqsubseteq D\} \tag{4.6}$$

Now $\Xi(KB_3)$ consists of the following clauses:

$$\neg A(x) \vee R(x, f(x)) \tag{4.7}$$

$$\neg A(x) \vee B(f(x)) \tag{4.8}$$

$$\neg B(x) \vee C(x) \tag{4.9}$$

$$D(x) \vee \neg R(x, y) \vee \neg C(y) \tag{4.10}$$

Assuming literals are ordered as $D(x) \succ C(x) \succ B(x) \succ A(x)$, by saturating $\Xi(KB_3)$ we obtain the following clauses (the notation $R(xx; yy)$ means that a clause is derived by resolving clauses $xx$ and $yy$):

$$\neg A(x) \vee D(x) \vee \neg C(f(x)) \qquad \text{R(4.7;4.10)} \tag{4.11}$$

$$\neg A(x) \vee D(x) \vee \neg B(f(x)) \qquad \text{R(4.11;4.9)} \tag{4.12}$$

$$\neg A(x) \vee D(x) \qquad \text{R(4.12;4.8)} \tag{4.13}$$

By eliminating clauses containing function symbols, we get $DD(KB_3)$ as follows:

$$C(x) \leftarrow B(x) \tag{4.14}$$

$$D(x) \leftarrow R(x, y), C(y) \tag{4.15}$$

$$D(x) \leftarrow A(x) \tag{4.16}$$

It is instructive to consider the role of each rule in $\mathsf{DD}(KB_3)$. Whereas the axiom $B \sqsubseteq C$ in $KB_3$ is applicable to all individuals in a model, the rule (4.14) is applicable only to named individuals. The relationship between $\exists R.C \sqsubseteq D$ and (4.15) is analogous. However, to compensate for the fact that (4.14) and (4.15) derive consequences only about named individuals, $\mathsf{DD}(KB_3)$ contains the rule (4.16), which is produced by the saturation of $\Xi(KB_\mathcal{T})$ by $\mathcal{R}_{DL}$. This rule can be thought of as a shortcut: instead of introducing for each $x$ in $A$ an $R$-successor $y$ in $B$, propagating $y$ to $C$, and then concluding that $x$ is in $D$, the rule (4.16) derives that all members of $A$ are members of $D$ in one step, and thus ensures that $\mathsf{DD}(KB_3)$ and $KB_3$ entail the same set of ground facts.

Finally, we take $KB_4$ to be the knowledge base introduced in Section 4.1. Note that the concept $\exists R.\exists R.A$ contains a nonatomic subconcept $\exists R.A$, to which we apply structural transformation, and replace it by a new atomic concept $Q$.[1] Hence, the set of clauses $\Xi(KB_4)$ consists of the following clauses:

$$\neg A(x) \vee R(x, f(x)) \tag{4.17}$$
$$\neg A(x) \vee A(f(x)) \tag{4.18}$$
$$Q(x) \vee \neg R(x, y) \vee \neg A(y) \tag{4.19}$$
$$B(x) \vee \neg R(x, y) \vee \neg Q(y) \tag{4.20}$$

Assuming literals are ordered as $Q(x) \succ B(x) \succ A(x)$, by saturating $\Xi(KB_4)$ using $\mathcal{R}_{DL}$ we obtain the following new clauses:

$$\neg A(x) \vee Q(x) \vee \neg A(f(x)) \quad \text{R(4.17;4.19)} \tag{4.21}$$
$$\neg A(x) \vee Q(x) \quad \text{R(4.21;4.18)} \tag{4.22}$$
$$\neg A(x) \vee B(x) \vee \neg Q(f(x)) \quad \text{R(4.17;4.20)} \tag{4.23}$$
$$\neg A(x) \vee B(x) \vee \neg A(f(x)) \quad \text{R(4.23;4.22)} \tag{4.24}$$
$$\neg A(x) \vee B(x) \quad \text{R(4.24;4.18)} \tag{4.25}$$

After eliminating clauses containing function symbols, we obtain $\mathsf{DD}(KB)$ as the following set of rules:

$$Q(x) \leftarrow R(x, y), A(y) \tag{4.26}$$
$$B(x) \leftarrow R(x, y), Q(y) \tag{4.27}$$
$$Q(x) \leftarrow A(x) \tag{4.28}$$
$$B(x) \leftarrow A(x) \tag{4.29}$$

The intuitive meaning behind these rules is somewhat obscured because we introduced a new predicate $Q$. However, we are not interested in ground consequences related to $Q$, and, since $Q$ is a new predicate, it cannot occur in any ABox one might consider in conjunction with $KB_4$. Hence, we can perform all possible resolution inferences with the $Q$ predicate in advance (that is, we can apply *rule unfolding* to $Q$), which yields the following datalog program:

$$B(x) \leftarrow A(x) \tag{4.30}$$
$$B(x) \leftarrow R(x, y), A(y) \tag{4.31}$$
$$B(x) \leftarrow R(x, y), R(y, z), A(z) \tag{4.32}$$

Now the intuitive meaning of these rules can be explained as follows. The rule (4.30) takes into account that each named individual that is in $A$ has a chain of at

[1] The rules (4.2)–(4.4) do not contain $Q$ because they are obtained by direct clausification, without applying structural transformation.

least two unnamed $R$-successors. The rule (4.31) takes into account that a named individual may be explicitly linked through $R$ to another individual in $A$ that then has at least one unnamed $R$-successor. Finally, the rule (4.32) takes into account that a named individual may be explicitly linked through an $R$-chain of length two to a named individual that is in $A$. Only $R$-chains of length two are considered, because $KB_4$ contains the concept $\exists R.\exists R.A$, which effectively checks for such $R$-chains.

## 4.7 Extending the Algorithms to $\mathcal{SHIQ}(\mathbf{D})$

The algorithms for reducing a $\mathcal{SHIQ}(\mathbf{D})$ knowledge base to a disjunctive datalog program, presented in the subsequent chapters, are based on the same ideas as the algorithms for $\mathcal{ALC}$. Next, we outline the issues that must be addressed to extend the basic algorithms:

- The most important difference is that $\mathcal{SHIQ}$ provides for number restrictions, which we translate into first-order logic with equality. This requires using basic superposition [14, 96] instead of ordered resolution. The decision procedure for $\mathcal{SHIQ}$ by basic superposition is derived along the same principles as for $\mathcal{ALC}$; however, due to more complex clause types and a more complex calculus, establishing closure under inferences is significantly more involved.

- As we discuss in Section 5.4, we were able to derive a decision procedure by basic superposition only for a slightly weaker logic $\mathcal{SHIQ}^-$. Namely, even though basic superposition is a fairly optimized calculus, it does not restrict the inferences enough to obtain a decision procedure for full $\mathcal{SHIQ}$. The problems are caused by an interaction of inverse properties, number restrictions, and role hierarchies. To solve these problems, we extend basic superposition with a new *decomposition* inference rule. We show that such a calculus is sound and complete, and that it decides $\mathcal{SHIQ}$.

- The reduction into disjunctive datalog is also more involved. Namely, due to number restrictions, the ground clauses derivable by basic superposition can contain literals of the form $A(f(a))$. It is now not possible to simply throw away all clauses containing function symbols after saturation. Instead, we encode a ground functional term $f(a)$ using a fresh constant $a_f$.

- To be able to handle datatypes, in Chapter 6 we first derive a general approach for reasoning with concrete domains in the framework of resolution. Then, we apply this approach to derive a decision procedure, and, consequently, the reduction algorithm for $\mathcal{SHIQ}(\mathbf{D})$.

- $\mathcal{SHIQ}$ provides for transitivity axioms, which, in their clausal form, raise a number of issues for resolution decision procedures [82]. Hence, in Section 5.2 we show how to eliminate transitivity axioms from knowledge bases without affecting satisfiablity, using a transformation similar to the ones from [144] and [129].

- A slight technical complication with clausifying $\mathcal{SHIQ}$ axioms arises because, even though a concept $D = \leq n\, R.A$ is in negation-normal form, the concept $A$ occurs in $D$ under negative polarity. Hence, the structural transformation for $\mathcal{SHIQ}$ is modified to take this into account.

## 4.8 Discussion

In this section we discuss some aspects of our algorithms that may seem surprising at the first glance.

### 4.8.1 Independence of the Reduction and the Query

Theorem 4.5.2 shows that the program $\mathsf{DD}(KB)$ is independent of the query, as long as the query is a positive atomic concept or a role.[2] Hence, $\mathsf{DD}(KB)$ can be computed once, and can be used to answer any number of atomic queries.

On the contrary, if the query involves a nonatomic concept $C$ (even if $C$ is a negated atomic concept), then query answering can be reduced to entailment of positive ground facts, by introducing a new name $Q$, and by adding the axiom $C \sqsubseteq Q$ to the TBox. Obviously, $\mathsf{DD}(KB \cup \{C \sqsubseteq Q\})$ may depend on the query concept $C$.

Intuitively, the reduction algorithm effectively derives all nonground facts following from the terminological part of a knowledge base. Moreover, a complex concept $C$, even if used only in the query, introduces terminological knowledge. Therefore, the reduction is independent of atomic query concepts, but depends on nonatomic ones.

We point out that, if $\mathsf{DD}(KB)$ is interpreted under standard first-order semantics, then $KB$ and $\mathsf{DD}(KB)$ also entail the same set of negative ground facts. However, disjunctive datalog programs are usually interpreted under minimal model semantics. As we discuss in the following subsection, minimal models do affect the entailment of negative ground facts; therefore, we restrict Theorem 4.5.2 to positive facts only.

### 4.8.2 Minimal vs. Arbitrary Models

Disjunctive datalog programs are usually interpreted under *minimal model* semantics: $P \models_c \alpha$ means that $\alpha$ is true in all minimal models of a program $P$, where minimality is defined w.r.t. set inclusion. In this way, disjunctive datalog implements a kind of *closed-world* semantics. On the contrary, description logics assume the standard first-order semantics: $KB \models \alpha$ means that $\alpha$ is true in all models of $KB$. Thus, description logics implement *open-world* semantics. It may come as a surprise that a logic whose semantics is defined for arbitrary models can be embedded in a logic that considers only minimal models. The answer is found in two important observations: (*i*) our reduction produces only positive datalog programs (that is, programs without negation-as-failure), and (*ii*) the distinction between first-order and minimal-model semantics is not relevant for certain types of questions.

[2] Note that the algorithm for full $\mathcal{SHIQ}$ allows only simple roles in queries.

Assume that $P$ is a positive datalog program and that $\alpha$ is a positive ground atom. Then, $P \models \alpha$ if and only if $P \models_c \alpha$. Namely, if $\alpha$ is true in each model of $P$, it is true in each minimal model of $P$ as well, and vice versa. Therefore, for entailment of positive ground atoms, it is not important whether the semantics of $P$ is defined w.r.t. minimal or w.r.t. general first-order models.

Assume now that $\alpha$ is a negative ground atom. Then, the difference between minimal model semantics and first-order semantics is relevant, as shown by the following example. For $\alpha = \neg A(b)$ and $P = \{A(a)\}$, it is clear that $P \not\models \alpha$. Namely, $\neg A(b)$ is not explicitly derivable from the facts in $P$: $M_1 = \{A(a), A(b)\}$ is a first-order model of $P$, and $\alpha$ is false in $M_1$. However, $P$ has exactly one minimal model $M_2 = \{A(a)\}$, and $\neg A(b)$ is obviously true in $M_2$, so $P \models_c \alpha$.

The choice of semantics also affects subsumption. For $\alpha = \forall x : [C(x) \rightarrow D(x)]$ and $P = \{C(a), D(a)\}$, we have $P \not\models \alpha$: $M_1 = \{C(a), D(a), C(b)\}$ is a first-order model of $P$, in which $\alpha$ is false. On the contrary, the only minimal model of $P$ is $M_2 = \{C(a), D(a)\}$, and $\alpha$ is true in $M_2$, so $P \models_c \alpha$. The distinction between minimal and general first-order models fundamentally changes the computational properties of concept subsumption: equivalence of general programs under minimal model semantics is undecidable [136] whereas, under first-order semantics, it is decidable and can be reduced to satisfiability checking using standard transformations.

To summarize, the difference between first-order and minimal model semantics is not relevant for answering positive queries in positive datalog programs; however, it is relevant for queries that involve negation or for concept subsumption. Theorem 4.5.2 reflects this: it does not say that minimal models of $\mathsf{DD}(KB)$ are somehow related to $KB$; it simply says that all positive ground facts entailed by $KB$ are contained in each minimal model of $\mathsf{DD}(KB)$ and vice versa.

It would therefore be incorrect to check whether $KB \models \neg A(a)$ by checking if $\mathsf{DD}(KB) \models_c \neg A(a)$. To solve this task using our algorithms, we must reduce it to entailment of positive ground facts: for $NotA$ a new concept, $KB \models \neg A(a)$ if and only if $\mathsf{DD}(KB \cup \{\neg A \sqsubseteq NotA\}) \models_c NotA(a)$.

Similarly, it would be incorrect to check whether $KB \models C \sqsubseteq D$ by checking whether $\mathsf{DD}(KB) \models_c \forall x : [C(x) \rightarrow D(x)]$. To solve the task using our algorithms, we again must reduce it to entailment of positive ground facts. If $C$ and $D$ are atomic concepts, $KB \models C \sqsubseteq D$ if and only if $\mathsf{DD}(KB) \cup \{C(\iota)\} \models_c D(\iota)$, where $\iota$ is a new individual not occurring in $KB$.

### 4.8.3 Complexity

The complexity of cautious query answering in a nonground positive disjunctive datalog program $P$ is co-NExpTime [42], due to the following reasons. Let $c$ be the number of constants, $v$ the maximal number of variables per rule, $p$ the number of predicates, and $a$ the maximal arity of a predicate in $P$; in general, all of these parameters are linear in $|P|$. Since $P$ is assumed to be a positive program, query answering in $P$ can be reduced to checking unsatisfiability of $P$ in the usual way. To decide satisfiability of $P$,

we compute the ground program $P^G = \mathsf{ground}(P)$ by replacing, in each rule, at most $v$ variables with one of the $c$ constants. This gives $c^v$ combinations, so $|P^G|$ is exponential in $|P|$. Furthermore, the number of ground atoms in $P^G$ is also exponential in $|P|$, as it is bounded by $p \cdot c^a$: each of the $c$ constants can occur at each of the $a$ positions in a ground atom in $P^G$. Now satisfiability of $P^G$ can be performed by guessing an interpretation $I$ for $P^G$, and then checking whether $I$ is a model of $P^G$. The first step requires choosing a truth value for each ground atom of $P^G$; this can obviously be performed in time exponential in $|P|$. The second step can be performed by checking the truth value of each rule of $P^G$; since the length of the rules is linear in $|P|$, but the number of rules is exponential in $|P|$, this step can also be performed in time exponential in $|P|$. Since the algorithm is nondeterministic, these two steps together give NExpTime complexity of satisfiability checking, and co-NExpTime complexity of unsatisfiability checking and query answering.

Note that the results in [42] actually give co-NExpTime$^{\mathrm{NP}}$as the complexity of query answering. This is because the authors consider a more general case of disjunctive datalog programs with negation-as-failure under stable model semantics. In such a case, it does not suffice to find an arbitrary model; one must additionally check if the model is minimal, which can be performed using an NP oracle. For positive datalog programs without negation-as-failure this is not needed: if we find a model, we know that a minimal model exists as well.

These results suggest that reducing DLs to disjunctive datalog might increase the complexity of reasoning. We show, however, that the previous calculation overestimates the complexity in case of $\mathsf{DD}(KB)$. Let $P^G_{DL} = \mathsf{ground}(\mathsf{DD}(KB))$. Now the number of variables in a rule from $\mathsf{DD}(KB)$ is at most two, so $|P^G_{DL}|$ is still exponential in $|KB|$. Furthermore, the arity of literals in $\mathsf{DD}(KB)$ is at most two, so the number of ground literals in $P^G_{DL}$ is polynomial in $|KB|$. Hence, an interpretation $I_{DL}$ for $P^G_{DL}$ can be guessed in time polynomial in $|KB|$, and all such interpretations can be examined in time exponential in $|KB|$. Checking if $I_{DL}$ is a model of $P^G_{DL}$ can be performed in time exponential in $|KB|$. These two steps combined give us the ExpTime complexity for satisfiability checking, and also for unsatisfiability checking and query answering. To summarize, even though $|\mathsf{DD}(KB)|$ is exponential in $|KB|$, query answering can be performed in time exponential in $|KB|$ because $(i)$ the length of the rules in $\mathsf{DD}(KB)$ is polynomial in $|KB|$, and $(ii)$ the arity of the literals is bounded.

### 4.8.4 Descriptive vs. Minimal-Model Semantics

Our reduction preserves the so-called descriptive semantics. Namely, Nebel has shown that knowledge bases containing terminological cycles are not definitorial [95]: for a fixed partial interpretation of atomic concepts, several interpretations of nonatomic concepts may exist. In such a case, it might be reasonable to designate a particular interpretation as the intended one, with least and greatest fixpoint models being the obvious candidates. However, Nebel argues that it is not clear which interpretation best matches the intuition, as choosing either of the fixpoint models has its drawbacks.

Consequently, most description logic systems implement the descriptive semantics, which coincides with that of Definition 3.1.2.

By Theorem 4.5.2, our decision procedure implements exactly the descriptive semantics. Namely, $\mathsf{DD}(KB)$ entails exactly those ground facts that are derivable using the resolution decision procedure, and the latter implements the descriptive semantics.

### 4.8.5 Unique Name Assumption

The semantics of $\mathcal{ALC}$ and $\mathcal{SHIQ}(\mathbf{D})$ does not require different symbols to be interpreted as different objects, whereas the standard semantics of disjunctive datalog does require it. Therefore, it may seem surprising that a logic without unique name assumption (UNA) can be embedded into a logic that strictly requires it.

To understand why our algorithms are correct, note that UNA can be used to derive new consequences from a theory only if the theory employs equality. For example, $\mathcal{ALC}$ provides neither number restrictions, nor explicit individual equality statements, so it actually does not require equality reasoning. Given a knowledge base $KB$, to enforce UNA we can append an axiom $a_i \not\approx a_j$ for each $a_i \neq a_j$. However, since $KB$ does not contain the equality predicate, these inequality axioms do not participate in any inference with $\mathcal{ALC}$-clauses, so they are not needed in the first place. A model-theoretic explanation is given by the following claim, which can easily be proved for logics without any form of equality:[3] for each model $I$, where distinct constants are interpreted by the same objects, there is a model $I'$ where all constants are interpreted by distinct objects. To summarize, in the case of a logic such as $\mathcal{ALC}$, we simply do not care whether either $KB$ or $\mathsf{DD}(KB)$ employs UNA, as this does not change the entailed set of facts.

The situation changes slightly for logics such as $\mathcal{SHIQ}$, which require a form of equality reasoning. Consider $KB = \{\top \sqsubseteq \leq 1\, R,\, R(a,b),\, R(a,c)\}$. Without UNA, $KB$ is satisfiable, and $KB \models b \approx c$. Namely, the first axiom requires $R$ to be functional, so $b$ and $c$ must be interpreted as the same object. On the contrary, with UNA, $KB$ is unsatisfiable.

It is easy to see that $\mathsf{DD}(KB)$ consists of the rules (4.33)–(4.35). Note that $\mathsf{DD}(KB)$ is now a disjunctive datalog program with equality, which is quite different from the type of equality usually considered in disjunctive datalog.

$$y_1 \approx y_2 \leftarrow R(x, y_1), R(x, y_2) \tag{4.33}$$

$$R(a, b) \tag{4.34}$$

$$R(a, c) \tag{4.35}$$

In [42], the authors consider disjunctive datalog in which equality atoms can occur in rule bodies under negation-as-failure. Such programs are interpreted under UNA, so an inequality atom $\neg(x \approx y)$ actually checks whether $x$ and $y$ are bound to syntactically different individuals.

[3] Note that number restrictions typically require equality.

In our case, however, we allow equality to occur not only in the rule bodies, but also in the rule heads. This means we can actually *derive* two individuals to be equal. Such inferences are not directly supported in disjunctive datalog; however, they can be simulated using the well-known encoding from [46]. The main idea is to treat $\approx$ as an ordinary predicate, for which the properties of equality are explicitly axiomatized. Hence, for a disjunctive datalog program $P$ with equality, we compute the program $P_{\approx}$ that states that $\approx$ is reflexive, symmetric, and transitive, and contains the following rule for each distinct predicate $R$ from $P$ and a position $i$ in $R$:

$$R(x_1, \ldots, y_i, \ldots, x_n) \leftarrow R(x_1, \ldots, x_i, \ldots, x_n), x_i \approx y_i \tag{4.36}$$

The program $P \cup P_{\approx}$ is now a disjunctive datalog program in which $\approx$ is just another predicate, so we are free to interpret $P \cup P_{\approx}$ with or without UNA, just as we discussed previously. Furthermore, in each Herbrand model $I_{\approx}$ of $P \cup P_{\approx}$, the rules in $P_{\approx}$ ensure that $\approx$ is interpreted as a congruence relation, so we can always transform $I_{\approx}$ to an interpretation $I$ by replacing each individual $a$ with the equivalence class of $\approx$ to which $a$ belongs. The resulting interpretation $I$ is a model of $P$ [46].

Observe that the encoding sketched in the previous paragraph is polynomial in the size of $P$, so it does not increase the worst-case complexity of reasoning. However, it is known that the replacement rules do introduce problems in practice. Therefore, in Section 7.5 we present a slight optimization of this technique, which explores the idea of the basic superposition to reduce the number of required replacement rules.

To summarize, $\mathsf{DD}(KB)$ can freely be interpreted under UNA, as long as we keep the following issues in mind. If $\mathsf{DD}(KB)$ does not contain equality, we do not care about UNA, as we cannot tell the difference. Otherwise, $\mathsf{DD}(KB)$ can be interpreted under UNA if we treat $\approx$ as a normal predicate whose properties are explicitly axiomatized. Finally, note that $\mathsf{DD}(KB)$ does not contain negation-as-failure. Nonmonotonic reasoning without UNA is an open problem; however, it is not relevant in our case.

### 4.8.6 The Size of $\mathsf{DD}(KB)$

By Theorem 4.5.2, the number of the rules of $|\mathsf{DD}(KB)|$ can be exponential in $|KB|$, which may seem discouraging in practice. We address this issue in two ways.

On the theoretical side, in Chapter 8 we observe that this blowup is only in the size of the TBox of $KB$. Hence, a sufficiently large ABox may actually dominate the size of the rules. This observation leads to novel *data complexity* results: in general, checking satisfiability of $KB$ is NP-complete in $|KB_{\mathcal{A}}|$; furthermore, we identify a new fragment of $\mathcal{SHIQ}(\mathbf{D})$ for which satisfiability checking is even polynomial in $|KB_{\mathcal{A}}|$.

On the practical side, in Section 7.3 we present an important optimization that allows us to remove many rules from $|\mathsf{DD}(KB)|$. Intuitively, the saturation of $\Xi(KB)$ introduces clauses that are entailed by other clauses. Such clauses are needed to derive all relevant clauses without function symbols. However, after saturation, the rules in $\mathsf{DD}(KB)$ that are entailed by other rules in $\mathsf{DD}(KB)$ can freely be removed. Our experiments show that this drastically reduces the size of $\mathsf{DD}(KB)$.

### 4.8.7 The Benefits of Reducing DLs to Disjunctive Datalog

We see two main benefits of our approach. On the theoretical side, our reduction makes it easier to derive novel data complexity results that we present in Chapter 8. Namely, the distinction between combined and data complexity has traditionally been studied in the context of (deductive) databases. Furthermore, we believe that our work sheds new light on the relationship between description logics and deductive databases, pointing out similarities and differences between the two knowledge representation formalisms. For example, as we extend description logics with rules in Chapter 9, we show that the rules can simply be appended to the result of the reduction; a result that is conceptually very easy to grasp.

On the practical side, the reduction to disjunctive datalog allows applying the magic sets transformation [18, 55, 34] and other optimization techniques, which are independent of the query answering algorithm, and can be reused as-is. Furthermore, in Section 7.6 we present an alternative technique for answering queries in datalog programs. If the program is not disjunctive, this technique falls back to the standard fixpoint saturation, which has been efficiently implemented in practice. Our algorithms thus follow the principle of *graceful degradation*: one pays a performance penalty only for features that one actually uses.

# Chapter 5

# Deciding $\mathcal{SHIQ}$ by Basic Superposition

In this chapter we present a resolution-based decision procedure for $\mathcal{SHIQ}$, which we then use in Chapter 7 to derive an algorithm for reducing a $\mathcal{SHIQ}$ knowledge base to a disjunctive datalog program.

Decision procedures based on clausal calculi have been derived for numerous fragments of first-order logic (an overview is given in Section 5.6). However, $\mathcal{SHIQ}$ cannot be directly embedded into any of these fragments. An exception is the two-variable fragment with counting quantifiers; however, our attempts to derive the reduction to disjunctive datalog using the decision procedure from [110] were unsuccessful. Hence, we designed a new, worst-case optimal decision procedure for $\mathcal{SHIQ}$, which itself has several novel aspects.

It is well known that the combination of inverse roles and counting quantifiers is difficult to handle algorithmically. On the model-theoretic side, such a combination of constructs easily causes a logic to lose the finite-model property; on the proof-theoretic side, it makes the tableau decision procedures more complex, by requiring sophisticated pair-wise blocking techniques to ensure termination [73]. This combination of constructs makes a resolution-based decision procedure more complex as well: contrary to most existing decision procedures for related logics (such as the ones presented in [47, 129]), to decide $\mathcal{SHIQ}$ it is necessary to consider clauses containing terms of depth two. To block certain inferences on such clauses, a calculus for equality stronger than superposition [9] is needed, so we use *basic superposition* [14, 96].

## 5.1 Decision Procedure Overview

The fundamental principles for deciding a first-order fragment $\mathcal{L}$ by resolution are known from [80]. First, one selects a sound and complete resolution calculus $\mathcal{C}$. Second, one identifies the set of clauses $\mathcal{N}_{\mathcal{L}}$ such that $\mathcal{N}_{\mathcal{L}}$ is finite for a finite signature, and that clausifying each formula $\varphi \in \mathcal{L}$ produces only clauses from $\mathcal{N}_{\mathcal{L}}$. Third, one

demonstrates that $\mathcal{N}_\mathcal{L}$ is *closed* under $\mathcal{C}$; that is, one shows that applying an inference of $\mathcal{C}$ to clauses from $\mathcal{N}_\mathcal{L}$ produces a clause in $\mathcal{N}_\mathcal{L}$. This is sufficient to obtain a refutation decision procedure for $\mathcal{L}$, since, in the worst case, $\mathcal{C}$ will derive all clauses of $\mathcal{N}_\mathcal{L}$.

A minor problem in deciding satisfiability of a $\mathcal{SHIQ}$ knowledge base $KB$ is caused by transitivity axioms, as they produce clauses without *covering literals*—literals containing all variables of a clause [76]. As shown in [82], termination of resolution on such clauses is very difficult to achieve. To address this, we show in Section 5.2 how to eliminate transitivity axioms by polynomially encoding a $\mathcal{SHIQ}$ knowledge base $KB$ into an equisatisfiable $\mathcal{ALCHIQ}$ knowledge base $\Omega(KB)$. After this initial transformation step, we consider only $\mathcal{ALCHIQ}$ knowledge bases.

A significantly more complex problem is that basic superposition alone decides only the $\mathcal{ALCHIQ}^-$ fragment of $\mathcal{ALCHIQ}$, in which number restrictions are allowed only on roles not having subroles. Namely, the combination of role hierarchies, inverse roles, and number restrictions may cause a saturation by basic superposition to produce terms of ever increasing depth, thus preventing termination. We address this problem in two stages.

In Section 5.3 we present a procedure for deciding satisfiability of an $\mathcal{ALCHIQ}^-$ knowledge base $KB$. We start by preprocessing $KB$ into a set of closures $\Xi(KB)$ as explained in Subsection 5.3.1. It is not difficult to see that $\Xi(KB)$ contains closures only of a syntactic form as in Table 5.1. We then saturate $\Xi(KB)$ under $\mathcal{BS}_{DL}$ with eager application of redundancy elimination rules, where $\mathcal{BS}_{DL}$ is the $\mathcal{BS}$ calculus parameterized according to Definition 5.3.3. We denote the saturated set of closures by $\mathsf{Sat}(\Xi(KB))$. Since $\mathcal{BS}_{DL}$ is sound and complete [14], $\mathsf{Sat}(\Xi(KB))$ contains the empty closure if and only if $\Xi(KB)$ is unsatisfiable. To obtain a decision procedure, we show that saturation always terminates. This is done in a proof-theoretic way along the lines outlined at the beginning of this section:

- We generalize the types of closures from Table 5.1 to $\mathcal{ALCHIQ}^-$-closures, which are presented in Table 5.2. In Lemma 5.3.4 we show that each closure occurring in $\Xi(KB)$ is an $\mathcal{ALCHIQ}^-$-closure.
- In Lemma 5.3.6 we show that, in any $\mathcal{BS}_{DL}$-derivation starting from $\Xi(KB)$, each inference produces either an $\mathcal{ALCHIQ}^-$-closure, or a closure that is redundant (and can therefore be deleted).
- In Lemma 5.3.9 we show that, for a finite knowledge base, the set of possible $\mathcal{ALCHIQ}^-$-closures occurring in any $\mathcal{BS}_{DL}$-derivation is finite.
- Termination is a simple consequence of these two lemmata: in the worst case, all possible $\mathcal{ALCHIQ}^-$-closures are derived in a saturation, after which all further inferences are redundant. The bound on the number of $\mathcal{ALCHIQ}^-$-closures yields the complexity of the algorithm, as demonstrated in Theorem 5.3.10.

To handle $\mathcal{ALCHIQ}$, in Subsection 5.4.1 we extend the basic superposition calculus with a new *decomposition* inference rule that transforms certain closures with

undesirable terms into several simpler closures. We show that decomposition is sound and complete. Furthermore, in Subsection 5.4.2 we show that it guarantees the termination of basic superposition for $\mathcal{ALCHIQ}$. It turns out that decomposition is quite a general and versatile rule: in Subsection 5.4.3 we use it to decide a slightly stronger logic $\mathcal{ALCHIQ}b$ that allows certain *safe* role expressions.

## 5.2 Eliminating Transitivity Axioms

In this section, we show how to eliminate transitivity axioms from a $\mathcal{SHIQ}$ knowledge base $KB$ by transforming it polynomially into an equisatisfiable $\mathcal{ALCHIQ}$ knowledge base $\Omega(KB)$. Since $\Omega(KB)$ is satisfiable if and only if $KB$ is, in the remaining sections we restrict our attention to $\mathcal{ALCHIQ}$ knowledge bases without loss of generality. For a $\mathcal{SHIQ}^-$ knowledge base $KB$, $\Omega(KB)$ is an $\mathcal{ALCHIQ}^-$ knowledge base, so we do not consider very simple roles explicitly in the definition of $\Omega$.

The transformation presented here is similar to the algorithm for transforming $\mathcal{SHIQ}$ concepts to $\mathcal{ALCIQ}b$ concepts presented in [144] ($\mathcal{ALCIQ}b$ does not provide for a role hierarchy, but allows certain types of Boolean operations on roles). A similar transformation for encoding formulae of the modal logic K4 (that is, the propositional modal logic with a transitive modality) into formulae of the modal logic K (that is, the propositional modal logic with an unrestricted modality) was presented in [129].

**Definition 5.2.1.** *For a $\mathcal{SHIQ}$ knowledge base $KB$, let $\mathsf{clos}(KB)$ denote the* concept closure *of $KB$, defined as the smallest set of concepts satisfying the following conditions:*

- *If $C \sqsubseteq D \in KB_{\mathcal{T}}$, then $\mathsf{NNF}(\neg C \sqcup D) \in \mathsf{clos}(KB)$;*
- *If $C \equiv D \in KB_{\mathcal{T}}$, then $\mathsf{NNF}(\neg C \sqcup D) \in \mathsf{clos}(KB)$ and $\mathsf{NNF}(\neg D \sqcup C) \in \mathsf{clos}(KB)$;*
- *If $C(a) \in KB_{\mathcal{A}}$, then $\mathsf{NNF}(C) \in \mathsf{clos}(KB)$;*
- *If $C \in \mathsf{clos}(KB)$ and $D$ is a subconcept of $C$, then $D \in \mathsf{clos}(KB)$;*
- *If $\leq n\, R.C \in \mathsf{clos}(KB)$, then $\mathsf{NNF}(\neg C) \in \mathsf{clos}(KB)$;*
- *If $\forall R.C \in \mathsf{clos}(KB)$, $S \sqsubseteq^* R$, and $\mathsf{Trans}(S) \in KB_{\mathcal{R}}$, then $\forall S.C \in \mathsf{clos}(KB)$.*

Note that all concepts in $\mathsf{clos}(KB)$ are in NNF. We now define the operator $\Omega$:

**Definition 5.2.2.** *For a $\mathcal{SHIQ}$ knowledge base $KB$, $\Omega(KB)$ is an $\mathcal{ALCHIQ}$ knowledge base constructed as follows:*

- *$\Omega(KB)_{\mathcal{R}}$ is obtained from $KB_{\mathcal{R}}$ by removing all axioms $\mathsf{Trans}(R)$;*
- *$\Omega(KB)_{\mathcal{T}}$ is obtained by adding to $KB_{\mathcal{T}}$ the axiom $\forall R.C \sqsubseteq \forall S.(\forall S.C)$, for each concept $\forall R.C \in \mathsf{clos}(KB)$ and role $S$ such that $S \sqsubseteq^* R$ and $\mathsf{Trans}(S) \in KB_{\mathcal{R}}$;*
- *$\Omega(KB)_{\mathcal{A}} = KB_{\mathcal{A}}$.*

The encoding is polynomial in $|KB|$: the number of concepts in $\mathsf{clos}(KB)$ generated by a concept $C$ is bounded by $2 \cdot |C| \cdot |N_R|$, and, for each concept from $\mathsf{clos}(KB)$, we generate at most $|N_R|$ axioms in $\Omega(KB)_{\mathcal{T}}$. Next, we show that it does not affect satisfiability.

**Theorem 5.2.3.** *$KB$ is satisfiable if and only if $\Omega(KB)$ is satisfiable.*

*Proof.* ($\Rightarrow$) Assume that $I$ is a model of $KB$, but not of $\Omega(KB)$; that is, $\Omega(KB)$ contains an axiom that is not satisfied in $I$. Since $\Omega(KB)_{\mathcal{R}} \subseteq KB_{\mathcal{R}}$ and $KB_{\mathcal{T}} \subseteq \Omega(KB)_{\mathcal{T}}$, such an axiom must have been added in the second point of Definition 5.2.2. Hence, there is a domain element $\alpha$ such that $\alpha \in (\forall R.C)^I$, but $\alpha \notin (\forall S.(\forall S.C))^I$. There are two possibilities:

- There is no domain element $\beta$ for which $(\alpha, \beta) \in S^I$. Then $\alpha \in (\forall S.X)^I$, regardless of $X$. Hence, $\alpha \in (\forall S.(\forall S.C))^I$ holds, which is a contradiction.
- There is a domain element $\beta$ such that $(\alpha, \beta) \in S^I$. There are two possibilities:
  - If no domain element $\gamma$ exists such that $(\beta, \gamma) \in S^I$, then $\beta \in (\forall S.C)^I$.
  - If there is $\gamma$ such that $(\beta, \gamma) \in S^I$, by transitivity of $S$ we have $(\alpha, \gamma) \in S^I$. Since $S^I \subseteq R^I$ and $\alpha \in (\forall R.C)^I$, we have $\gamma \in C^I$. Since this holds for any $\gamma$, we have $\beta \in (\forall S.C)^I$.

  Either way, for any $\beta$, we have $\beta \in (\forall S.C)^I$, so $\alpha \in (\forall S.(\forall S.C))^I$, which is a contradiction.

($\Leftarrow$) As explained in Section 3.1, without loss of generality we can consider only knowledge bases with acyclic RBoxes. Let $I$ be a model of $\Omega(KB)$, and $I'$ an interpretation constructed from $I$ as follows:

- $\triangle^{I'} = \triangle^I$;
- For each individual $a$, $a^{I'} = a^I$;
- For each atomic concept $A \in \mathsf{clos}(KB)$, $A^{I'} = A^I$;
- If $\mathsf{Trans}(R) \in KB_{\mathcal{R}}$, then $R^{I'} = (R^I)^+$;
- If $\mathsf{Trans}(R) \notin KB_{\mathcal{R}}$, then $R^{I'} = R^I \cup \bigcup_{S \sqsubseteq^* R, S \neq R} S^{I'}$.

Since we assume that $KB_{\mathcal{R}}$ is acyclic, the induction is well-founded. By construction, $I'$ satisfies all transitivity axioms in $KB_{\mathcal{R}}$. Furthermore, $I'$ satisfies each inclusion axiom in $KB_{\mathcal{R}}$: if $R$ is not transitive, this is obvious from the construction; otherwise, this follows because $A^+ \cup B^+ \subseteq (A \cup B)^+$ for all binary relations $A$ and $B$. For each role $R$, we have $R^I \subseteq R^{I'}$; furthermore, if $R$ is simple, then $R^{I'} = R^I$.

For concepts $C$ and $D$ from $\mathsf{clos}(KB)$, let $C \lessdot D$ if and only if $C$ or $\mathsf{NNF}(\neg C)$ occur in $D$. We now show by induction on $\lessdot$ that, for each $D \in \mathsf{clos}(KB)$, $D^I \subseteq D^{I'}$. The

relation $\prec$ is obviously acyclic, so the induction is well-founded. For the base case, where $D$ is an atomic concept $A$ or a negation of an atomic concept $\neg A$, the claim follows immediately from the definition of $I'$. For the induction step, we examine possible forms that $D$ might have.

- For $D = C_1 \sqcap C_2$, assume that $\alpha \in (C_1 \sqcap C_2)^I$ for some $\alpha$. Then, $\alpha \in C_1^I$ and $\alpha \in C_2^I$. By the induction hypothesis, $\alpha \in C_1^{I'}$ and $\alpha \in C_2^{I'}$, so $\alpha \in (C_1 \sqcap C_2)^{I'}$.
- For $D = C_1 \sqcup C_2$, assume that $\alpha \in (C_1 \sqcup C_2)^I$ for some $\alpha$. If $\alpha \in C_1^I$, by the induction hypothesis, we have $\alpha \in C_1^{I'}$; if $\alpha \in C_2^I$, by the induction hypothesis, we have $\alpha \in C_2^{I'}$. Either way, $\alpha \in (C_1 \sqcup C_2)^{I'}$.
- For $D = \exists R.C$, assume that $\alpha \in (\exists R.C)^I$ for some $\alpha$. Then, $\beta$ exists such that $(\alpha, \beta) \in R^I$ and $\beta \in C^I$, so, by the induction hypothesis, $\beta \in C^{I'}$. Since $R^I \subseteq R^{I'}$, we have $(\alpha, \beta) \in R^{I'}$, so $\alpha \in (\exists R.C)^{I'}$.
- For $D = \forall R.C$, assume that $\alpha \in (\forall R.C)^I$ for some $\alpha$. If there is no object $\beta$ such that $(\alpha, \beta) \in R^{I'}$, then $\alpha \in (\forall R.C)^{I'}$. Otherwise, assume that such $\beta$ exists. There are two possibilities:
  - $(\alpha, \beta) \in R^I$. Then, $\beta \in C^I$, so, by the induction hypothesis, $\beta \in C^{I'}$.
  - $(\alpha, \beta) \notin R^I$. Then, a role $S \sqsubseteq^* R$ with $\mathsf{Trans}(S) \in KB_{\mathcal{R}}$ exists, as well as a path $(\alpha, \gamma_1) \in S^I$, $(\gamma_1, \gamma_2) \in S^I$, …, $(\gamma_n, \beta) \in S^I$ with $n \geq 0$. But then, $\forall R.C \sqsubseteq \forall S.(\forall S.C) \in \Omega(KB)_{\mathcal{T}}$, so $\alpha \in (\forall S.(\forall S.C))^I$ and $\gamma_1 \in (\forall S.C)^I$. Furthermore, $\forall S.C \sqsubseteq \forall S.(\forall S.C) \in \Omega(KB)_{\mathcal{T}}$, so $\gamma_i \in (\forall S.C)^I$ for $0 \leq i \leq n$. For $i = n$, we get that $\beta \in C^I$, so, by the induction hypothesis, $\beta \in C^{I'}$.

  For any $\beta$, in both cases we have that $\beta \in C^{I'}$, so we conclude that $\alpha \in (\forall R.C)^{I'}$.
- For $D = \geq n\, R.C$, assume that $\alpha \in (\geq n\, R.C)^I$. Then, there are at least $n$ distinct domain elements $\beta_i$ such that $(\alpha, \beta_i) \in R^I$ and $\beta_i \in C^I$. By the induction hypothesis, we have $\beta_i \in C^{I'}$, and, because $R^I \subseteq R^{I'}$, we have $\alpha \in (\geq n\, R.C)^{I'}$.
- For $D = \leq n\, R.C$, because $R$ is simple, we have $R^I = R^{I'}$. Let $E = \mathsf{NNF}(\neg C)$. Assume now that $\alpha \in (\leq n\, R.C)^I$, but $\alpha \notin (\leq n\, R.C)^{I'}$. Then, $\beta$ exists such that $(\alpha, \beta) \in R^I$, $\beta \notin C^I$, $\beta \in C^{I'}$, $\beta \in E^I$, and $\beta \notin E^{I'}$. However, since $E \in \mathsf{clos}(KB)$, by induction hypothesis we have $\beta \in E^{I'}$, which is a contradiction. Hence, $\alpha \in (\leq n\, R.C)^{I'}$.

It is now obvious that each ABox axiom of the form $C(a)$ from $KB$ is satisfied in $I'$: since $I$ is a model of $\Omega(KB)$, we have $a^I \in C^I$, but since $C^I \subseteq C^{I'}$, we have $a^{I'} \in C^{I'}$ as well. Also, any ABox axiom of the form $R(a, b)$ from $KB$ is satisfied in $I'$: since $I$ is a model of $\Omega(KB)$, we have $(a^I, b^I) \in R^I$, but since $R^I \subseteq R^{I'}$, we have $(a^{I'}, b^{I'}) \in R^{I'}$. For an ABox axiom of the form $\neg S(a, b)$, $S$ must be a simple role, so $S^I = S^{I'}$, and $(a^I, b^I) \notin S^I$ implies $(a^{I'}, b^{I'}) \notin S^{I'}$. Finally, any TBox axiom of the form $C \subseteq D$ from

$KB$ is satisfied in $I'$: since $I$ is a model of $\Omega(KB)$, we have that $\triangle^I \subseteq (\neg C \sqcup D)^I$, but since $(\neg C \sqcup D)^I \subseteq (\neg C \sqcup D)^{I'}$, we have $\triangle^{I'} \subseteq (\neg C \sqcup D)^{I'}$. Similar arguments hold for any TBox axiom of the form $C \equiv D$. □

Note that, for $\alpha$ of the form $(\neg)A(a)$ with $A$ an atomic concept, or of the form $(\neg)S(a,b)$ with $S$ a simple role, $\Omega(KB \cup \{\alpha\}) = \Omega(KB) \cup \{\alpha\}$, so $KB \models \alpha$ if and only if $\Omega(KB) \models \alpha$. Hence, transitivity axioms can be eliminated once, and the resulting knowledge base can be used for query answering. Unfortunately, the models of $KB$ and $\Omega(KB)$ may differ in the interpretation of complex roles. Therefore, $\Omega(KB)$ cannot be used to answer ground queries where $S$ is a complex role. This is why, in Definition 3.1.1, we restrict negative role membership axioms to simple roles only.

## 5.3 Deciding $\mathcal{ALCHIQ}^-$

In this section, we present an algorithm for deciding satisfiability of an $\mathcal{ALCHIQ}^-$ knowledge base $KB$ by basic superposition.

### 5.3.1 Preprocessing

To decide satisfiability of $KB$, we must transform it into a clausal form. As discussed in Section 4.3, a straightforward transformation of $\pi(KB)$ into disjunctive normal form might exponentially increase the formula size and could destroy the structure of the formula. Therefore, before clausification, we apply to $KB$ the *structural transformation* [108, 99, 8], also known as *renaming*.

**Definition 5.3.1.** *Let $C$ be a concept and $\Lambda$ a function assigning to $C$ the set of positions $p \neq \epsilon$ such that $C|_p$ is not a literal concept and, for all positions $q$ below $p$, $C|_q$ is a literal concept. Then, $\mathsf{Def}(C)$ is defined recursively as follows:*

- *If $\Lambda(C) = \emptyset$, then $\mathsf{Def}(C) = \{C\}$;*
- *If $\Lambda(C) \neq \emptyset$, then choose $p \in \Lambda(C)$ and let $\mathsf{Def}(C)$ be as follows, where $Q$ is a new globally unique atomic concept:*

$$\mathsf{Def}(C) = \begin{cases} \{\neg Q \sqcup C|_p\} \cup \mathsf{Def}(C[Q]_p) & \text{if } \mathsf{pol}(C,p) = 1 \\ \{Q \sqcup \neg C|_p\} \cup \mathsf{Def}(C[Q]_p) & \text{if } \mathsf{pol}(C,p) = -1 \end{cases}$$

*The operator $\mathsf{Cls}$ for clausifying a first-order formula, defined in Section 2.1, is extended to concepts as follows:*

$$\mathsf{Cls}(C) = \bigcup_{D \in \mathsf{Def}(C)} \mathsf{Cls}(\forall x : \pi_y(D, x))$$

*For an extensionally reduced $\mathcal{ALCHIQ}$ knowledge base $KB$, $\Xi(KB)$ is the smallest set of closures satisfying the following conditions:*

- *For each abstract role name* $R \in N_{R_a}$, $\mathsf{Cls}(\pi(R)) \subseteq \Xi(KB)$;
- *For each RBox or ABox axiom* $\alpha$ *in* $KB$, $\mathsf{Cls}(\pi(\alpha)) \subseteq \Xi(KB)$;
- *For each TBox axiom* $C \sqsubseteq D$ *in* $KB$, $\mathsf{Cls}(\neg C \sqcup D) \subseteq \Xi(KB)$;
- *For each TBox axiom* $C \equiv D$ *in* $KB$, $\mathsf{Cls}(\neg C \sqcup D) \subseteq \Xi(KB)$ *and* $\mathsf{Cls}(\neg D \sqcup C) \subseteq \Xi(KB)$.

*If* $KB$ *is not extensionally reduced, then* $\Xi(KB) = \Xi(KB')$*, where* $KB'$ *is an extensionally reduced knowledge base obtained from* $KB$ *as explained in Section 3.1.*

The following lemma summarizes the properties of $\Xi(KB)$:

**Lemma 5.3.2.** *For* $KB$ *an* $\mathcal{ALCHIQ}^-$ *knowledge base, the following claims hold:*

1. *$KB$ is satisfiable if and only if $\Xi(KB)$ is satisfiable.*
2. *$\Xi(KB)$ can be computed in time polynomial in $|KB|$ for unary coding of numbers in input.*
3. *Each closure from $\Xi(KB)$ is of one of the types given in Table 5.1.*

*Proof.* (Claim 1.) It is easy to see that $\psi_C = \bigwedge_{D \in \mathsf{Def}(C)} \forall x : \pi_y(D, x)$ is actually the definitional normal form of $\varphi = \forall x : \pi_y(C, x)$ with respect to the set of positions of all nonatomic subformulae of $\varphi$. By the definitions of $\pi$ and $\mathsf{Cls}$, and because transformation into definitional normal form does not affect satisfiability [108, 99, 8], $KB$ and $\Xi(KB)$ are equisatisfiable.

(Claim 2.) The inductive step of $\mathsf{Def}(C)$ is applied at most once for each subconcept of $C$, so the number of new concepts $Q$ introduced by $\mathsf{Def}$ is linear in $|C|$, and $\mathsf{Def}(C)$ can be computed in polynomial time. For each $D \in \mathsf{Def}(C)$, the number of function symbols $f$ introduced by skolemizing $\forall x : \pi_y(D, x)$ is bounded by the maximal number occurring in a number restriction in $D$. For unary coding of numbers, $f$ is linear in $|D|$, so $\mathsf{Cls}(D)$ can be computed in time polynomial in $|D|$.

(Claim 3.) Observe that $\mathsf{Def}(C)$ contains only concepts of the form $D \sqcup \bigsqcup L_i$, where $D$ is a $\mathcal{SHIQ}$ concept containing only literal subconcepts and $L_i$ are literal concepts. By the definition of $\pi$ from Table 3.1, closures of type 1 axiomatize inverse properties; closures of type 2 correspond to role inclusion axioms; closures of types 3 and 4 are obtained for $D = \exists R.C$; closures of types 3, 4, and 5 are obtained for $D = \geq n\, R.C$; closures of type 6 are obtained for $D = \bigsqcup L_i$ or $D = \bigsqcap L_i$; closures of type 7 are obtained for $D = \forall R.C$ or $D = \leq n\, R.C$; finally, closures of types 8–11 correspond to ABox axioms. □

Using binary coding, a number $n$ can be represented using $\lceil \log_2 n \rceil$ bits, so the number of function symbols introduced by skolemization is exponential in the size of the input. Hence, for binary coding of numbers, translation into first-order logic incurs an exponential blowup.

Table 5.1: Closure Types after Preprocessing

| | |
|---|---|
| 1 | $\neg R(x,y) \vee \mathsf{Inv}(R)(y,x)$ |
| 2 | $\neg R(x,y) \vee S(x,y)$ |
| 3 | $\bigvee(\neg)C_i(x) \vee R(x,f(x))$ |
| 4 | $\bigvee(\neg)C_i(x) \vee (\neg)D(f(x))$ |
| 5 | $\bigvee(\neg)C_i(x) \vee f_i(x) \not\approx f_j(x)$ |
| 6 | $\bigvee(\neg)C_i(x)$ |
| 7 | $\bigvee(\neg)C_i(x) \vee \bigvee_{i=1}^{n} \neg R(x,y_i) \vee \bigvee_{i=1}^{n} D(y_i) \vee \bigvee_{i=1}^{n} \overset{n}{\underset{j=i+1}{}} y_i \approx y_j$ |
| 8 | $(\neg)C(a)$ |
| 9 | $(\neg)R(a,b)$ |
| 10 | $a \approx b$ |
| 11 | $a \not\approx b$ |

### 5.3.2 Parameters for Basic Superposition

To understand the following definition, it is necessary to keep in mind that literals $P(t_1,\ldots,t_n)$ are encoded as $P(t_1,\ldots,t_n) \approx \mathsf{T}$, as discussed in Section 2.5. Due to this encoding, predicate symbols become $\mathcal{E}$-function symbols, and atoms become $\mathcal{E}$-terms.

**Definition 5.3.3.** *Let $\mathcal{BS}_{DL}$ denote the $\mathcal{BS}$ calculus parameterized as follows:*

- *The $\mathcal{E}$-term ordering $\succ$ is a lexicographic path ordering induced by a total precedence $>$ over function, constant, and predicate symbols such that $f > c > P > \mathsf{T}$, for each function symbol $f$, constant symbol $c$, and predicate symbol $P$.*
- *The selection function selects every negative binary literal in each closure $C \cdot \sigma$.*

We show in Subsection 5.3.3 that, in applying $\mathcal{BS}_{DL}$ to $\Xi(KB)$, we need to compare only $\mathcal{E}$-terms occurring in closures of types 3–6 and 8 from Table 5.2. These closures contain at most one variable, so any LPO is total on $\mathcal{E}$-terms from such closures. This allows us to use a simpler extension of the $\mathcal{E}$-term ordering to literals than the one obtained by the two-fold multiset extension as explained in Section 2.5. We assign to each literal $L = s \circ t$ with $\circ \in \{\approx, \not\approx\}$ the triple $c_L = (\max(s,t), p_L, \min(s,t))$, where $(i)$ $\max(s,t)$ is the larger of the two $\mathcal{E}$-terms; $(ii)$ $p_L = 1$ if $\circ$ is $\not\approx$; $(iii)$ $p_L = 0$ if $\circ$ is $\approx$; and $(iv)$ $\min(s,t)$ is the smaller of the two $\mathcal{E}$-terms. Then, $L_1 \succ L_2$ if and only if $c_{L_1} \succ c_{L_2}$, where $c_{L_i}$ are compared lexicographically. The $\mathcal{E}$-term ordering is used to compare the first and the third position of $c_L$ whereas, for the second position, we take $1 \succ 0$. It is easy to see that this definition is equivalent to the one based on the two-fold multiset extension.

Ordering and selection constraints in $\mathcal{BS}$ are checked *a posteriori*—that is, after computing the unifier. This is more general, because some $\mathcal{E}$-terms may be comparable only after unification. For example, $s = f(x)$ and $t = y$ are not comparable under any

LPO. For $\sigma = \{x \mapsto a, y \mapsto g(f(a))\}$, however, we have $t\sigma \succ s\sigma$. The drawback of a posteriori checking of ordering constraints is that the unifier is often computed in vain, just to determine that constraints are not satisfied. However, LPOs are total on $\mathcal{E}$-terms from closures 3–6 and 8, so we can check ordering and selection constraints *a priori*—that is, before computing the unifier. Namely, if $s$ and $t$ are two $\mathcal{E}$-terms to be compared, they are either both ground or both have the same, single free variable, so they are always comparable by an LPO. Also, if $s \succ t$, then $s\sigma \succ t\sigma$ for each substitution $\sigma$, because LPOs are stable under substitutions.

### 5.3.3 Closure of $\mathcal{ALCHIQ}^-$-Closures under Inferences by $\mathcal{BS}_{DL}$

We now generalize the types of closures from Table 5.1 to $\mathcal{ALCHIQ}^-$-closures, presented in Table 5.2. For a term $t$, with $\mathbf{P}(t)$ we denote a possibly empty disjunction of the form $(\neg)P_1(t) \vee \ldots \vee (\neg)P_n(t)$. With $\mathbf{P}(\mathbf{f}(x))$ we denote a possibly empty disjunction of the form $\mathbf{P_1}(f_1(x)) \vee \ldots \vee \mathbf{P_m}(f_m(x))$. Note that this definition allows each $\mathbf{P_i}(f_i(x))$ to contain positive and negative literals. With $\langle t \rangle$ we denote that the term $t$ may, but need not be marked. In all closure types, the disjuncts $\mathbf{P}(t)$ or $\mathbf{P}(\mathbf{f}(x))$ may be empty. To distinguish the closures of types 5 and 6, we assume that each closure of type 6 always contains at least one term of the form $f(g(x))$. Finally, with $\approx/\not\approx$ we denote a positive or a negative equality literal.

**Lemma 5.3.4.** *Each closure from $\Xi(KB)$ is of a type from Table 5.2. Furthermore, for each function symbol $f$ occurring in $\Xi(KB)$, there is exactly one closure of type 3 containing $f(x)$ unmarked; this closure is called the $R^f$-*generator*, the disjunction of unary literals in this closure is denoted with $\mathbf{P^f}(x)$ and is called the $f$-*support*, and $R$ is called the* designated role for *$f$ and is denoted with $\mathsf{role}(f)$.*

*Proof.* The first claim follows trivially from Lemma 5.3.2. For the second claim, observe that each closure of type 3 is generated by skolemizing an existentially quantified subformula. Since each skolemization introduces a fresh function symbol, this symbol is associated with exactly one closure of type 3. □

Next, we prove a lemma that shows which literals can be maximal in closures of types 3, 4, 5, 6, and 8 under the ordering and the selection function of $\mathcal{BS}_{DL}$.

**Lemma 5.3.5.** *The maximal literal of an $\mathcal{ALCHIQ}^-$-closure that participates in an inference by $\mathcal{BS}_{DL}$ satisfies the following conditions:*

- *In a closure of type 3, the literal $R(x, \langle f(x) \rangle)$ is always maximal.*
- *In a closure of type 4, the literal $R([f(x)], x)$ is always maximal.*
- *In a closure of type 5, a literal $(\neg)P(x)$ can be maximal only if the closure does not contain a term $f(x)$.*
- *In a closure of type 6, only a literal containing a term $f([g(x)])$ can be maximal.*

Table 5.2: Types of $\mathcal{ALCHIQ}^-$-Closures

| | |
|---|---|
| 1 | $\neg R(x,y) \vee \mathsf{Inv}(R)(y,x)$ |
| 2 | $\neg R(x,y) \vee S(x,y)$ |
| 3 | $\mathbf{P^f}(x) \vee R(x, \langle f(x) \rangle)$ |
| 4 | $\mathbf{P^f}(x) \vee R([f(x)], x)$ |
| 5 | $\mathbf{P_1}(x) \vee \mathbf{P_2}(\langle \mathbf{f}(x) \rangle) \vee \bigvee \langle f_i(x) \rangle \approx/\not\approx \langle f_j(x) \rangle$ |
| 6 | $\mathbf{P_1}(x) \vee \mathbf{P_2}([g(x)]) \vee \mathbf{P_3}(\langle \mathbf{f}([g(x)]) \rangle) \vee \bigvee \langle t_i \rangle \approx/\not\approx \langle t_j \rangle$<br>where $t_i$ and $t_j$ are either of the form $f([g(x)])$ or of the form $x$ |
| 7 | $\mathbf{P_1}(x) \vee \bigvee_{i=1}^{n} \neg R(x, y_i) \vee \bigvee_{i=1}^{n} \mathbf{P_2}(y_i) \vee \bigvee_{i=1}^{n} \overset{n}{\underset{j=i+1}{}} y_i \approx y_j$ |
| 8 | $\mathbf{R}(\langle \mathbf{a} \rangle, \langle \mathbf{b} \rangle) \vee \mathbf{P}(\langle \mathbf{t} \rangle) \vee \bigvee \langle t_i \rangle \approx/\not\approx \langle t_j \rangle$<br>where $t$, $t_i$, and $t_j$ are either a constant $b$ or a term $f_i([a])$ |

**Conditions:**

- $(i)$: In any term $f(t)$, the inner term $t$ occurs marked.
- $(ii)$: In all positive equality literals with at least one function symbol, both sides are marked.
- $(iii)$: Any closure that contains a term $f(t)$, contains $\mathbf{P^f}(t)$ as well.
- $(iv)$: In a literal $[f_i(t)] \approx [f_j(t)]$, $\mathsf{role}(f_i) = \mathsf{role}(f_j)$.
- $(v)$: In a literal $[f(g(x))] \approx x$, $\mathsf{role}(f) = \mathsf{Inv}(\mathsf{role}(g))$.
- $(vi)$: For each $[f_i(a)] \approx [b]$ in a closure $C$, a *witness* closure of $C$ exists that is of form $R(\langle a \rangle, \langle b \rangle) \vee D$, $\mathsf{role}(f_i) = R$, $D$ does not contain function symbols or negative binary literals, and $D$ is contained in $C$.

- *In a closure of type 8, a literal $(\neg)R(a,b)$, $(\neg)P(a)$, $a \approx b$, or $a \not\approx b$ can be maximal only if the closure does not contain a function symbol.*

*Proof.* For each term $t$, function symbol $f$, and predicate symbol $P$, since $f > P$ by Definition 5.3.3 and $f(t) \succ t$, we have $f(t) \succ P(t)$. Hence, $R(x, f(x)) \succ f(x) \succ P(x)$, so $R(x, f(x))$ is always maximal in a closure of type 3; the claim for a closure of type 4 follows analogously. Furthermore, $P''(f(g(x))) \succ f(g(x)) \succ P'(g(x)) \succ g(x) \succ P(x)$, thus implying the claims for closures of types 5 and 6. Finally, for each function symbol $f$, constants $a$, $b$, and $c$, unary predicate symbol $P$, and binary predicate symbol $R$, by Definition 5.3.3 we have $f(a) \succ P(b)$, $f(a) \succ R(b,c)$, and $f(a) \succ b$, thus implying the claim for a closure of type 8. □

We now prove the core result that our decision procedure is based upon.

**Lemma 5.3.6.** *Let $\Xi(KB) = N_0, \ldots, N_i \cup \{C\}$ be a $\mathcal{BS}_{DL}$-derivation, where $C$ is the conclusion derived from premises in $N_i$. Then, $C$ is either an $\mathcal{ALCHIQ}^-$-closure, or it is redundant in $N_i$.*

*Proof.* We prove the lemma by induction on the derivation length. By Lemma 5.3.4, $N_0$ contains only $\mathcal{ALCHIQ}^-$-closures, so the induction base holds. For the induction

step, we examine all possible applications of inference rules of $\mathcal{BS}_{DL}$ to closures in $N_i$. We first show that the conclusion has the structure of an $\mathcal{ALCHIQ}^-$-closure, and later prove that conditions $(i)$–$(vi)$ hold as well.

**Inferences with closures of types 1 and 2.** Since negative binary literals are always selected, and superposition into variables is not necessary, closures of types 1 and 2 can participate only as main premises in resolution inferences with closures of types 3, 4, and 8. Obviously, the unifier binds the variables $x$ and $y$ to corresponding terms in the positive premise, and the result is of type 3, 4, or 8.

**Inferences between closures of types 5, 6, and 8.** Consider any inference between closures of types 5 or 6 with free variables $x$ and $x'$, respectively. Since the term $g(x)$ in some $f([g(x)])$ is always marked, terms can be unified only at their root positions. The following pairs of terms from premises are unifiable:

- For $x$ and $x'$, $f(x)$ and $f(x')$, or $f([g(x)])$ and $f([g(x')])$, the unifier has the form $\sigma = \{x \mapsto x'\}$, and the conclusion is a closure of type 5 or 6. Note that superposition from $f(g(x)) \approx x$ into $f(g(x')) \not\approx x'$ results in $x' \not\approx x'$, which is eagerly eliminated by reflexivity resolution.
- For $x$ and $g(x')$, or $f(x)$ and $f([g(x')])$, the unifier is $\sigma = \{x \mapsto g(x')\}$, and the conclusion is a closure of type 5 or 6.
- For $x$ and $f(g(x'))$, the unifier is $\sigma = \{x \mapsto f(g(x'))\}$, and the conclusion is a closure of type 5 or 6.

Inferences between closures of type 6 and 8 are not possible, because a term of the form $f(g(x))$ does not unify with a term of the form $a$ or $f(a)$. For closures of types 5 and 8, the unifier is either $\sigma = \{x \mapsto a\}$ or $\sigma = \{x \mapsto f(a)\}$, and the conclusion is of type 8. Inferences between closures of type 8 have an empty unifier, so the conclusion is trivially of type 8.

**Inferences with a closure of type 7.** Since all binary literals are always selected, a closure of type 7 can participate only in a hyperresolution inference as the main premise $C$. The side premises can have the maximal literals of the form $R(x_i, \langle f_i(x_i)\rangle)$, $R([g_i(x_i)], x_i)$, or $R(\langle a\rangle, \langle b_i\rangle)$. These combinations are possible:

- Assume that there are two (or more) side premises with the maximal literal of the form $R([g_i(x_i)], x_i)$. Without loss of generality, these premises can be assigned indices 1 and 2. Since $g_1(x_1)$ and $g_1(x_2)$ are unified with $x$, we have $g_1 = g_2$. Moreover, $\sigma$ contains mappings $y_1 \mapsto x_1, y_2 \mapsto x_1$, and $x_2 \mapsto x_1$. Since $C$ contains a literal $y_i \approx y_j$ for each pair of indices $i$ and $j$, the conclusion contains $x_1 \approx x_1$, so it is a tautology.

- Assume that the first side premise has the maximal literal $R([g(x')], x')$. Since $g(x')$ does not unify with a constant, no side premise can be of type 8. The unifier $\sigma$ is of the form $\{x \mapsto g(x'), x_i \mapsto g(x'), y_1 \mapsto x', y_i \mapsto f_i(g(x'))$ for $2 \leq i \leq n\}$. If $n = 1$, the conclusion is of type 5; otherwise, it is of type 6.

- Assume that all side premises have the maximal literals $R(x_i, \langle f_i(x_i)\rangle)$. The unifier $\sigma$ is of the form $\{x_i \mapsto x, y_i \mapsto f_i(x)\}$, so the conclusion is of type 5.

- Assume that some side premises have the maximal literal $R(x_i, \langle f_i(x_i)\rangle)$, for $1 \leq i \leq k$, and $R(\langle a\rangle, \langle b_i\rangle)$ for $k < i \leq n$ (all ground premises must have the same first argument in the maximal literal, because all these arguments should unify with $x$). The unifier $\sigma$ contains mappings of the form $x \mapsto a, x_i \mapsto a, y_i \mapsto f_i(a)$ for $1 \leq i \leq k$, and $y_i \mapsto b_i$ for $k + 1 \leq i \leq n$, so the conclusion is of type 8.

**Superposition into a closure of type 3.** Consider superposition into a closure of type 3 with a free variable $x'$. By Lemma 5.3.5, $(w \approx v) \cdot \rho$ can only be the literal $R(x', f(x'))$, with $R$ being the designated role for $f$. There are these possibilities:

- $(C \vee s \approx t) \cdot \rho$ is a closure of type 5, 6, or 8 with $(s \approx t) \cdot \rho$ of the form $[f(u)] \approx [g(u)]$. The unifier is $\sigma = \{x' \mapsto u\}$, so the conclusion is $S = \mathbf{P^f}([u]) \vee R([u], [g(u)]) \vee C \cdot \rho$, where $\mathbf{P^g}([u]) \subseteq C \cdot \rho$. By Condition $(iv)$, the $R^g$-generator $P^g(y) \vee R(y, g(y))$ exists, and it subsumes $S$ by the substitution $\{y \mapsto u\}$.

- $(C \vee s \approx t) \cdot \rho$ is a closure of type 6 with $(s \approx t) \cdot \rho$ of the form $[f(g(x))] \approx x$. The unifier is $\sigma = \{x' \mapsto g(x)\}$, so the conclusion is $S = \mathbf{P^f}([g(x)]) \vee R([g(x)], x) \vee C \cdot \rho$, where $\mathbf{P^g}(x) \subseteq C \cdot \rho$. By Condition $(v)$, the closure $\mathbf{P^g}(y) \vee R([g(y)], y)$ exists, and it subsumes $S$ by the substitution $\{y \mapsto x\}$.

- $(C \vee s \approx t) \cdot \rho$ is a closure of type 8 with $(s \approx t) \cdot \rho$ of the form $[f(a)] \approx [b]$. The unifier is $\sigma = \{x' \mapsto a\}$, so the conclusion is $S = \mathbf{P^f}([a]) \vee R([a], [b]) \vee C \cdot \rho$. By Condition $(vi)$, a witness of the form $R(\langle a\rangle, \langle b\rangle) \vee D$, where $D \subseteq C \cdot \rho$, exists, and it subsumes $S$ by the empty substitution.

In all three cases, the subsuming closure might be subsumed by some other closure; however, subsumption is transitive, so this other closure then subsumes the inference conclusion. Hence, a superposition into a closure of type 3 is always redundant.

**Equality factoring.** Ordering constraints allow us to optimize the application of equality factoring. Only closures of types 5, 6, or 8 are candidates for equality factoring. The premise has the form $(C \vee s \approx t \vee s' \approx t') \cdot \rho$, where $s\rho \approx t\rho$ is maximal with respect to $C\rho \vee s'\rho \approx t'\rho$, $t\rho \not\succeq s\rho$, and $t'\rho \not\succeq s'\rho$. The unifier $\sigma$ is always empty. If we assume that simplification by duplicate literal elimination is applied eagerly, we safely conclude that $(s \approx t) \cdot \rho$ is strictly maximal, so $t'$ and $t$ cannot be $\mathsf{T}$. Hence, terms $s\rho$, $t\rho$, $s'\rho$ and $t'\rho$ are either all ground, or all contain the same free variable $x$. The

ordering $\succ$ is total on such terms, so the ordering constraints are equivalent to $t\rho \prec s\rho$ and $t'\rho \prec s'\rho$. Because $s\rho \approx t\rho$ is strictly maximal, we have $t\rho \succ t'\rho$.

Consider now the case where all equalities involved in the inference are marked, so $s$, $t$, $s'$, and $t'$ are variables. This is the case for all closures of types 5 and 6, and for some closures of type 8. The conclusion has the form $(C \vee t \not\approx t' \vee s' \approx t') \cdot \rho$, where $t$ is a variable and $t\rho \succ t'\rho$; such a conclusion is a basic tautology, which is redundant and can be deleted.

Hence, provided that duplicate literal elimination is applied eagerly, equality factoring is redundant for all closures, apart from closures of type 8, where it can be applied to equalities of the form $\langle a \rangle \approx \langle b \rangle$ in which at least one term is not marked. Depending on the marking, equality factoring either yields a basic tautology, which is redundant, or a closure of type 8. Note that a closure of type 8 can contain disjunctions of the form $R([a]\,, b) \vee R(a, [b])$, to which duplicate literal removal does not apply directly due to incompatible markers. However, we assume that in such a case the markers are eagerly retracted. Thus, such a disjunction is converted into $R(a, b) \vee R(a, b)$, which is then collapsed into $R(a, b)$.

**Reflexivity resolution.** Reflexivity resolution can only be applied to a closure of type 5, 6, and 8 with the empty unifier, so it produces a closure of type 5, 6, or 8. Since the unifier is always empty, the conclusion always subsumes the premise, so this inference rule should be applied eagerly.

**Conditions.** We now show that conditions from Table 5.2 hold for each nonredundant inference conclusion.

Condition $(i)$: If a closure $C$ satisfies Condition $(i)$, applying a unifier $\sigma$ to it only instantiates variables, so $C\sigma$ satisfies $(i)$ as well. Since no inference removes markers, Condition $(i)$ holds for any conclusion.

Condition $(ii)$: All positive equality literals with at least one function symbol are generated by hyperresolution with a closure of type 7, so all terms in positive equalities in the conclusion are marked. Since no inference removes markers from the roots of the terms occurring in equalities, Condition $(ii)$ holds for any conclusion.

Condition $(iii)$: If a closure $C$ contains $f([t])$ and satisfies Condition $(iii)$, by Lemma 5.3.5, literals from $\mathbf{P^f}([t])$ cannot participate in an inference. Furthermore, a variable is instantiated simultaneously in $f([t])$ and $P^f([t])$. The terms containing function symbols occurring in an inference conclusion always stem from one of the inference premises, so Condition $(iii)$ holds for any conclusion.

Conditions $(iv)$ and $(v)$: All equality literals containing function symbols are generated by hyperresolution with a main premise $C$ of type 7. Since the role $R$ occurring in $C$ is very simple, a closure of type 3 or 4 containing $R$ cannot be resolved with a closure of type 2. Hence, for all side premises of the form $\mathbf{P^f}(x) \vee R(x, \langle f(x) \rangle)$, we have $\mathsf{role}(f) = R$, and, for a side premise of the form $\mathbf{P^g}(x) \vee R([g(x)]\,, x)$, we have $\mathsf{role}(g) = \mathsf{Inv}(R)$. Hence, Conditions $(iv)$ and $(v)$ are satisfied for each conclusion of

a hyperresolution inference. Furthermore, by Condition (*ii*), superposition into positive equality literals containing function symbols is not possible, and, in each literal $[f_i(x)] \approx [f_j(x)]$, the variable $x$ is instantiated simultaneously. Hence, Conditions (*iv*) and (*v*) hold for any conclusion of any inference.

Condition (*vi*): All literals of the form $[f(a)] \approx [b]$ are generated by hyperresolution involving a side premise $E_1$ of type 8 with the maximal literal $R(\langle a \rangle, \langle b \rangle)$, and a side premise $E_2$ of type $\mathbf{P^f}(x) \vee R(x, \langle f(x) \rangle)$. Since $R$ occurring in such $C$ is very simple, a closure of type 8 cannot be resolved with a closure of type 2, so $\mathsf{role}(f) = R$. Since the literal $R(\langle a \rangle, \langle b \rangle)$ is maximal in $E_1$, by Lemma 5.3.5, no literal from $E_1$ contains a function symbol, and $E_2$ does not contain a negative binary literal. Hence, the conclusion satisfies Condition (*vi*). Assume now that Condition (*vi*) holds for some closure $C$, where $D$ is a witness of $[f(a)] \approx [b]$. Since each literal from $D$ neither contains a function symbol, nor is it a negative binary literal, by Lemma 5.3.5, no literal from $C$ occurring in $D$ can participate in an inference, so all literals are present in the conclusion. Finally, both sides of all equality literals containing function symbols are marked, so each equality literal containing function symbols derived in an inference must always occur in some of the premises. Hence, Condition (*vi*) holds for any conclusion. □

The following corollary follows from the proof of Lemma 5.3.6:

**Corollary 5.3.7.** *If a closure of type 8 participates in a $\mathcal{BS}_{DL}$ inference in a derivation from Lemma 5.3.6, the unifier $\sigma$ contains only ground mappings of the form $x \mapsto a$ and $x \mapsto f(b)$, and the conclusion is a closure of type 8. Furthermore, a closure of type 8 cannot participate in an inference with a closure of type 4 or 6.*

The following corollary is useful for optimizing the algorithm in Chapter 7.

**Corollary 5.3.8.** *Let $KB$ be an $\mathcal{ALCHIQ}^-$ knowledge base, containing neither at-most number restrictions occurring under positive, nor at-least number restrictions occurring under negative polarity. Then, in a saturation of $\Xi(KB)$ by $\mathcal{BS}_{DL}$, closures of type 8 do not contain function symbols. Furthermore, a closure of type 8 can participate in an inference only with closures not containing function symbols.*

*Proof.* Let $KB$ be as specified in the corollary, and let (*) denote the following property: for each closure $C$ of type 7 from $\Xi(KB)$, $C$ contains exactly one literal $\neg R(x, y)$ and it does not contain equality literals. We now prove the first claim by induction on the derivation length. The base case is obvious, since each ABox closure in $\Xi(KB) = N_0$ is of a type from Table 5.1, and it does not contain a function symbol. For the induction step, we consider all inferences generating a closure of type 8 in $N_{i+1}$. A closure with an equality literal containing a function symbol might be generated only by a hyperresolution with a closure of type 7 containing equality literals; however, this is not possible by (*). A literal $(\neg)C([g(a)])$ might be derived by hyperresolving a closure of type 7 with a side premise of type 3 and of type 8, but this is again not possible by (*). The only remaining possibility to derive a literal $(\neg)C([g(a)])$ is by superposition

from $[f(a)] \approx [g(a)]$ into $(\neg)C(f(x))$; however, this is not possible, since $N_i$ does not contain equality literals with function symbols.

For the second claim, note that, since a closure of type 8 does not contain a function symbol, it can participate in an inference only with a literal not containing a function symbol; since such a literal must be maximal, it can occur only in a closure not containing a function symbol. □

### 5.3.4 Termination and Complexity Analysis

We now show that the number of $\mathcal{ALCHIQ}^-$-closures is finite for a finite signature. This, in combination with Lemma 5.3.6 and the soundness and completeness of $\mathcal{BS}_{DL}$, shows that saturation of $\mathcal{BS}_{DL}$ with eager application of redundancy elimination rules decides satisfiability of $\mathcal{ALCHIQ}^-$ knowledge bases.

**Lemma 5.3.9.** *Let $N_i$ be any closure set encountered in a derivation from Lemma 5.3.6. If $C$ is a closure in $N_i$, then the number of literals in $C$ is at most polynomial in $|KB|$, for unary coding of numbers in input. Furthermore, $|N_i|$ is at most exponential in $|KB|$, for unary coding of numbers in input.*

*Proof.* By Lemma 5.3.6, $N_i$ can contain only $\mathcal{ALCHIQ}^-$-closures. Since redundant closures are eliminated eagerly, $N_i$ cannot contain closures with duplicate literals or closures identical up to variable renaming. Let $r$ denote the number of role predicates, $a$ the number of atomic concept predicates, $c$ the number of constants, and $f$ the number of function symbols occurring in the signature of $\Xi(KB)$. By definition of $|KB|$, $r$ and $c$ are obviously linear in $|KB|$. Furthermore, $a$ is also linear in $|KB|$, since the number of new atomic concept predicates introduced during preprocessing is bounded by the number of subconcepts of each concept, which is linear in $|KB|$. The number $f$ is bounded by the sum of all numbers $n$ in $\geq n\, R.C$ and $\leq n\, R.C$, plus one for each $\exists R.C$ and $\forall R.C$ occurring in $KB$. Since numbers are coded in unary, $f$ is linear in $|KB|$. Let $n$ denote the maximal number occurring in any number restriction; for unary coding of numbers, $n$ is linear in $|KB|$.

Consider now the maximal number of literals in a closure of type 5. The maximal number of literals for $\mathbf{P_1}(x)$ is $2a$ (factor 2 allows for each atomic concept predicate to occur positively or negatively), for $\mathbf{P_2}(\langle \mathbf{f}(x) \rangle)$ it is $2a \cdot 2f$ ($f$ is multiplied by 2 since each term may or may not be marked), for equalities it is $f^2$ (both terms are always marked), and for inequalities it is $4f^2$ (factor 4 allows for each side of the equality to be marked or not). Hence, the maximal number of literals is $2a + 4af + f^2 + 4f^2$. For a closure of type 6, the maximal number of literals is $2a + 2a + 4af + (f^2 + f) + (4f^2 + 2f)$: possible choices for $g$ do not contribute to the closure length, and the expressions in parenthesis take into account that each term in an equality or an inequality can be $f_i(g(x))$ or $x$. For a closure of type 8, the maximal number of literals is bounded by $2r \cdot 2c \cdot 2c + 2a \cdot 2c + 2a \cdot 2f \cdot c + 2 \cdot (4c^2 + c \cdot f^2 + cf \cdot 2c)$: the factor 2 in front of the parenthesis takes into account that equalities and inequalities can have the same form, and the expression in the parenthesis counts all possible forms these literals can

have. The maximal number of literals of closures of type 1 and 2 is obviously 2, and for closures of type 3 and 4 it is $a+1$. For a closure of type 7, the number of variables $y_i$ is bounded by $n$: each such closure in $\Xi(KB)$ contains at most $n$ variables, and no inference increases the number of variables. The maximal number of literals in a closure of type 7 is $n+4c+n^2$, since the choice for $R$ does not contribute to the closure length. Hence, the maximal number of literals in any closure in $N_i$ is polynomial in $|KB|$, for unary coding of numbers.

The maximal number of closures of types 1–6 and 8 in $N_i$ is now easily obtained as follows: if $C_\ell$ is the closure with the maximal number of literals $\ell$ for some closure type, then there are $2^\ell$ subsets of literals of $C_\ell$. To obtain the total number of closures, one must multiply $2^\ell$ with the number of closure-wide choices. For closures of type 6, the function symbol $g$ can be chosen in $f$ ways. For closures of type 3 and 4, one can choose $R$ and $f$ in $rf$ ways. For closures of type 1, one can choose $R$ in $r$ ways. For closures of type 2, one can choose $R$ and $S$ in $r^2$ ways. Since all these factors are polynomial in $|KB|$ for unary coding of numbers, we obtain an exponential bound on the number of closures of types 1–6 and 8. Finally, no inference derives a new closure of type 7, so $N_i$ contains only those closures of type 7 that are contained in $\Xi(KB)$. □

**Theorem 5.3.10.** *For an $\mathcal{ALCHIQ}^-$ knowledge base $KB$, saturation of $\Xi(KB)$ by $\mathcal{BS}_{DL}$ with eager application of redundancy elimination rules decides satisfiability of $KB$, and runs in time exponential in $|KB|$, for unary coding of numbers in input.*

*Proof.* Translation of $KB$ into $\Xi(KB)$ can be performed in time polynomial in $|KB|$ by Lemma 5.3.2, and $\Xi(KB)$ contains only $\mathcal{ALCHIQ}^-$-closures by Lemma 5.3.4. Let $\ell$ denote the maximal number of closures occurring in the closure set in a derivation as specified in Lemma 5.3.6, and let $l$ denote the maximal number of literals in a closure. By Lemma 5.3.9, $\ell$ is exponential, and $l$ is polynomial in $|KB|$, for unary coding of numbers. Since all terms are bounded in size, they can be compared in constant time, and a maximal term in a closure can be selected in time linear in $l$. A subsumption algorithm, running in time exponential in $l$, was presented in [53]. Furthermore, the subsumption check is performed at most for each pair of closures, so it takes at most exponential time. For closures other than of type 7, each of the four inference rules can potentially be applied to any closure pair, so, because exactly one literal of each closure is eligible for inference, we get at most $4\ell^2$ inferences. For a hyperresolution inference with a closure of type 7, $n$ side premises can be chosen in $\ell^n$ ways. Hence, the number of applications of inference rules of $\mathcal{BS}_{DL}$ is bounded by $4\ell^2+\ell^n$, which is exponential in $|KB|$, for unary coding of numbers in input. Now it is obvious that $\Xi(KB)$ is saturated after performing at most an exponential number of steps, after which the saturation terminates. Since $\mathcal{BS}_{DL}$ is sound and complete with eager application of redundancy elimination rules, the claim of the theorem follows. □

In the proof of Theorem 5.3.10, we assumed an exponential algorithm for checking subsumption. In practice, it is known that modern theorem provers spend up to 90

percent of their time in subsumption checking, so efficient subsumption checking is crucial for practical applicability of resolution theorem proving.

Fortunately, subsumption checks for $\mathcal{ALCHIQ}^-$-closures can be performed in polynomial time as follows. In [53], it was shown that subsumption between closures having at most one variable can be checked in polynomial time. This algorithm can be easily extended with an additional $\eta$-reducibility test, yielding a polynomial algorithm for checking subsumption of closures of type 3, 4, 5, 6, and 8. Checking whether a closure of type 1 or 2 subsumes some other closure can be performed by matching the negative literal first, and then checking whether the positive literal matches as well, which can be performed in quadratic time.

Let $C$ and $C'$ be two closures of type 7. For $C$ to subsume $C'$, the only possibility is that $\sigma$ contains mappings $y_i \mapsto y'_i$ and $x \mapsto x'$. Hence, $C$ subsumes $C'$ only if the number of variables $y_i$ in $C$ is smaller than in $C'$, both closures contain the same role $R$, and the predicates occurring in $\mathbf{P_1}(x)$ and $\mathbf{P_2}(\mathbf{y})$ of $C$ are a subset of the predicates occurring in $C'$, respectively. These checks can obviously be performed in polynomial time. Finally, a closure $C$ of type 5 can subsume a closure $C'$ of type 7 only if $C$ subsumes $\mathbf{P_1}(x)$ or $\mathbf{P_2}(y_i)$. These checks can be performed in polynomial time, because $C$, $\mathbf{P_1}(x)$, and $\mathbf{P_2}(y_i)$ contain only one variable.

## 5.4 Removing the Restriction to Very Simple Roles

In this section, we remove the restriction to very simple roles from the algorithm from Section 5.3, and thus obtain an algorithm for deciding satisfiability of an $\mathcal{ALCHIQ}$ knowledge base $KB$. Our main problem is that, during saturation of $\Xi(KB)$ by $\mathcal{BS}_{DL}$, we can obtain closures whose structure corresponds to Table 5.2, but for which conditions (*iii*)–(*vi*) do not hold; we call such closures $\mathcal{ALCHIQ}$-closures.

For example, let $KB$ be a knowledge base containing axioms (5.1)–(5.9). On the right-hand side, we show the translation of $KB$ into closures $\Xi(KB)$:

| | | |
|---|---|---|
| (5.1) | $R \sqsubseteq T$ | $\neg R(x,y) \vee T(x,y)$ |
| (5.2) | $S \sqsubseteq T$ | $\neg S(x,y) \vee T(x,y)$ |
| (5.3) | $C \sqsubseteq \exists R.\top$ | $\neg C(x) \vee R(x, f(x))$ |
| (5.4) | $\top \sqsubseteq \exists S^-.\top$ | $S^-(x, g(x))$ |
| (5.5) | $\top \sqsubseteq \leq 1\, T$ | $\neg T(x,y_1) \vee \neg T(x,y_2) \vee y_1 \approx y_2$ |
| (5.6) | $\exists S.\top \sqsubseteq D$ | $\neg S(x,y) \vee D(x)$ |
| (5.7) | $\exists R.\top \sqsubseteq \neg D$ | $\neg R(x,y) \vee \neg D(x)$ |
| (5.8) | $\top \sqsubseteq C$ | $C(x)$ |
| (5.9) | | $\neg S^-(x,y) \vee S(y,x)$ |

Consider a saturation of $\Xi(KB)$ by $\mathcal{BS}_{DL}$ (the notation $R(xx; yy)$ means that a closure is derived by resolving closures $xx$ and $yy$):

$$S([g(x)], x) \quad \text{R(5.4;5.9)} \tag{5.10}$$

$$\neg C(x) \vee T(x, [f(x)]) \quad \text{R(5.1;5.3)} \tag{5.11}$$

$$T([g(x)], x) \quad \text{R(5.2;5.10)} \tag{5.12}$$

$$\neg C([g(x)]) \vee [f(g(x))] \approx x \quad \text{R(5.5;5.11;5.12)} \tag{5.13}$$

For the closure (5.13), condition $(v)$ from Table 5.2 is not satisfied. Namely, we have $\mathsf{role}(f) = R \neq \mathsf{Inv}(\mathsf{role}(g)) = \mathsf{Inv}(S^-) = S$; this is because a number restriction was stated in (5.5) on a role that is not very simple. Now (5.13) can be superposed into (5.3), resulting in (5.14):

$$\neg C([g(x)]) \vee R([g(x)], x) \tag{5.14}$$

Since condition $(v)$ is not satisfied for (5.13), we cannot assume that a closure subsuming (5.14) exists, as we did in the proof of Lemma 5.3.6. Hence, we must keep (5.14), which is obviously not an $\mathcal{ALCHIQ}$-closure. This might cause termination problems since, in general, (5.14) might be resolved with some closure of type 6 of the form $C([g(h(x))])$, producing a closure of the form $R([g(h(x))], [h(x)])$. The term depth in a binary literal is now two; resolving such a closure with a closure of type 7 can yield closures with even deeper terms. Hence, Lemma 5.3.9, stating that the number of closures that can be derived is finite, does not hold any more, so saturation does not necessarily terminate.

We did not find a way to refine the ordering or the selection function that would solve this problem. Furthermore, (5.14) is necessary for completeness. Namely, $KB$ is unsatisfiable, and the empty closure is derived by the following derivation, which involves (5.14):

$$D([g(x)]) \quad \text{R(5.6;5.10)} \tag{5.15}$$

$$\neg D([g(x)]) \vee \neg C([g(x)]) \quad \text{R(5.7;5.14)} \tag{5.16}$$

$$\neg C([g(x)]) \quad \text{R(5.15;5.16)} \tag{5.17}$$

$$\square \quad \text{R(5.8;5.17)} \tag{5.18}$$

### 5.4.1 Transformation by Decomposition

To solve the termination problems for $\mathcal{ALCHIQ}$, in this subsection we introduce *decomposition*—a transformation that can be applied to the result of certain $\mathcal{BS}$ inferences. This transformation is generally applicable, and is not limited to description logic reasoning. We show that decomposition can be combined with basic superposition, but in a similar way one can show that it can be combined with any clausal calculus compatible with the standard notion of redundancy [13].

In the following, for $\mathbf{x}$ a vector of distinct variables $x_1, \ldots, x_n$ and $\mathbf{t}$ a vector of (not necessarily distinct) terms $t_1, \ldots, t_n$, let $\{\mathbf{x} \mapsto \mathbf{t}\}$ denote the substitution $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$, and let $Q([\mathbf{t}])$ denote $Q([t_1], \ldots, [t_n])$.

**Definition 5.4.1.** *Let $C \cdot \rho$ be a closure and $N$ a set of closures. A* decomposition of $C \cdot \rho$ w.r.t. $N$ *is a pair of closures $C_1 \cdot \rho \vee Q([\mathbf{t}])$ and $C_2 \cdot \theta \vee \neg Q(\mathbf{x})$, where $\mathbf{t}$ is a vector of $n$ terms, $\mathbf{x}$ is a vector of $n$ distinct variables, $n \geq 0$, satisfying these conditions:*

- $C = C_1 \cup C_2$;
- $\rho = \theta\{\mathbf{x} \mapsto \mathbf{t}\}$;
- $\mathbf{x}$ *is exactly the set of free variables of* $C_2\theta$;
- *If* $C_2 \cdot \theta \vee \neg Q'(\mathbf{x}) \in N$, *then* $Q = Q'$; *otherwise,* $Q$ *is a new predicate not occurring in* $N$.

*The closure $C_2 \cdot \theta$ is called the* fixed *part, the closure $C_1 \cdot \rho$ is called the* variable *part, and the predicate $Q$ is called the* definition predicate. *An application of decomposition is written as follows:*

$$C \cdot \rho \rightsquigarrow \begin{array}{l} C_1 \cdot \rho \ \vee \ Q([\mathbf{t}]) \\ C_2 \cdot \theta \ \vee \ \neg Q(\mathbf{x}) \end{array}$$

*Let $\xi$ be a $\mathcal{BS}$ inference with a most general unifier $\sigma$, a side premise $D_s \cdot \eta$, and a main premise $D_m \cdot \eta$; furthermore, let $L_m \cdot \eta$ be the literal from $D_m \cdot \eta$ on which the inference takes place. The conclusion of $\xi$ is* eligible for decomposition *if, for each ground substitution $\tau$ such that $\xi\tau$ satisfies the ordering constraints of $\mathcal{BS}$, we have $\neg Q(\mathbf{t})\tau \prec L_m\eta\sigma\tau$. With $\mathcal{BS}^+$ we denote the $\mathcal{BS}$ calculus where decomposition can be applied to conclusions of eligible inferences.*

As an example, consider superposition from a closure $[f(g(x))] \approx [h(g(x))]$ into a closure $C(x) \vee R(x, f(x))$, resulting in a closure $C([g(x)]) \vee R([g(x)], [h(g(x))])$. This is obviously not an $\mathcal{ALCHIQ}$-closure, so performing further inferences with it might prevent a saturation from terminating. However, this closure can be decomposed into $C([g(x)]) \vee Q_{R,f}([g(x)])$ and $\neg Q_{R,f}(x) \vee R(x, [h(x)])$, which are both $\mathcal{ALCHIQ}$-closures, so they do not cause termination problems. The inference is eligible for decomposition if we ensure that $\neg Q_{R,f}([g(x)]) \prec R(g(x), f(g(x))) = L_m\eta\sigma$; this can be done by using an LPO with a precedence in which $R > Q_{R,f}$.

The following lemma demonstrates that decomposition is a sound inference rule.

**Lemma 5.4.2.** *Let $N_0, \ldots, N_i$ be a $\mathcal{BS}^+$-derivation, and let $I_0$ be a model of $N_0$. For $i > 1$, let $I_i$ be an interpretation such that, if the inference deriving $N_i$ from $N_{i-1}$ involves a decomposition step introducing a new definition predicate $Q$, then $I_i = I_{i-1} \cup \{Q(\mathbf{s}) \mid \mathbf{s}$ is a vector of ground terms such that $C_2\theta\{\mathbf{x} \mapsto \mathbf{s}\}$ is true in $I_{i-1}\}$; otherwise $I_i = I_{i-1}$. Then, $I_i$ is a model of $N_i$.*

*Proof.* The proof is by induction on the length of the derivation $N_0, \ldots, N_i$. The base case is trivial, since $I_0$ is a model of $N_0$ by the assumption. For the induction step, we assume that $N_{i-1}$ has a model $I_{i-1}$ satisfying the conditions of the lemma, and consider possible inferences deriving $N_i$. For inferences without a decomposition step,

the claim is easy. Consider a positive superposition inference from $(s \approx t \vee C) \cdot \rho$ into $(w \approx v \vee D) \cdot \rho$ with a unifier $\sigma$, resulting in $(C \vee D \vee w[t]_p \approx v) \cdot \theta$, where $\theta = \rho\sigma$, and let $\tau$ be a ground substitution. If $(s \approx t) \cdot \theta\tau$ is false in $I_{i-1}$, then $C \cdot \theta\tau$ is true in $I_{i-1}$; if $(w \approx v) \cdot \theta\tau$ is false in $I_{i-1}$, then $D \cdot \theta\tau$ is true in $I_{i-1}$; and if both $(s \approx t) \cdot \theta\tau$ and $(w \approx v) \cdot \theta\tau$ are true in $I_{i-1}$, then $(w[t]_p \approx v) \cdot \theta\tau$ is true in $I_{i-1}$. The cases for other inference rules are analogous.

If the inference deriving $N_i$ from $N_{i-1}$ involves a decomposition step, then two cases are possible. If the predicate $Q$ is new, then $I_{i-1}$ is extended to $I_i$ by adding those ground literals $Q(\mathbf{s})$ for which $C_2\theta\{\mathbf{x} \mapsto \mathbf{s}\}$ is true in $I_{i-1}$. Hence, for each ground substitution $\tau$, in $C_2 \cdot \theta\tau \vee \neg Q(\mathbf{x})\tau$, either $C_2 \cdot \theta\tau$ or $\neg Q(\mathbf{x})\tau$ is true in $I_i$. Furthermore, if $C \cdot \rho\tau$ is true in $I_{i-1}$, then $C_1 \cdot \rho\tau \vee Q([\mathbf{t}])\tau$ is obviously true in $I_i$. If the predicate $Q$ is not new, then $I_i = I_{i-1}$. Then, by the induction hypothesis $Q(\mathbf{s})$ is true if and only if $C_2\theta\{\mathbf{x} \mapsto \mathbf{s}\}$ is true, and $C_1 \cdot \rho\tau \vee Q([\mathbf{t}])\tau$ is true in $I_i$ as in the previous case. □

We next show that decomposition is compatible with the standard notion of redundancy. This is the key step in showing completeness of $\mathcal{BS}^+$.

**Lemma 5.4.3.** *Let $\xi$ be a $\mathcal{BS}$ inference applied to premises from a closure set $N$, resulting in a closure $C \cdot \rho$. If $\xi$ is eligible for decomposition of $C \cdot \rho$ into $C_1 \cdot \rho \vee Q([\mathbf{t}])$ and $C_2 \cdot \theta \vee \neg Q(\mathbf{x})$, and the two latter closures are both redundant in $N$, then $\xi$ is redundant in $N$ as well.*

*Proof.* Let $\xi$ be an inference on a literal $L_m \cdot \eta$ from a main premise $D_m \cdot \eta$ and a side premise $D_s \cdot \eta$, with a most general unifier $\sigma$, resulting in a closure $C \cdot \rho$. Furthermore, let $R$ be a rewrite system and $\tau$ a ground substitution such that $\xi\tau$ satisfies the ordering constraints of $\mathcal{BS}$, and it is a variable irreducible ground instance of $\xi$ w.r.t. $R$. Finally, let $E_1 = (C_1 \cdot \rho \vee Q([\mathbf{t}]))\tau$ and $E_2 = (C_2 \cdot \theta \vee \neg Q(\mathbf{x}))\{\mathbf{x} \mapsto \mathbf{t}\}\tau$. Note that $D = \max(D_m \cdot \eta\sigma\tau, D_s \cdot \eta\sigma\tau) = D_m \cdot \eta\sigma\tau$.

By the ordering constraints of $\mathcal{BS}$ inference rules, $D_s\eta\sigma\tau \prec L_m\eta\sigma\tau$. Furthermore, superposition inferences are allowed only from the maximal side of an equality, so the inference always produces a literal $L'\eta\sigma\tau \prec L_m\eta\sigma\tau$. Finally, because $\xi$ is eligible for decomposition, $\neg Q(\mathbf{t})\tau \prec L_m\eta\sigma\tau$. Thus, if a literal $L\eta\sigma\tau \succ L_m\eta\sigma\tau$ occurs $n$ times in $E_1 \cup E_2$, it also occurs $n$ times in $D$. In other words, both $E_1$ and $E_2$ contain at most those literals larger than $L_m\eta\sigma\tau$ that also occur in $D$. All other literals in $E_1$ or $E_2$ are smaller than $L_m\eta\sigma\tau$. Since $L_m\eta\sigma\tau \in D$, we conclude that $E_1 \prec D$ and $E_2 \prec D$.

The vector of terms $\mathbf{t}$ is "extracted" from the substitution part of $C \cdot \rho$. Hence, if a term $t$ occurs in $E_1$ and $E_2$ at a substitution position, then $t$ occurs in $C \cdot \rho\tau$ also at a substitution position. Therefore, if $\xi\tau$ is variable irreducible w.r.t. $R$, so is $C \cdot \rho\tau$, and so are $E_1$ and $E_2$.

To summarize, for all rewrite systems $R$ and all ground substitutions $\tau$ such that $\xi\tau$ is a variable irreducible ground instance of $\xi$ w.r.t. $R$, the closures $E_1$ and $E_2$ are smaller than $D$, and are variable irreducible w.r.t $R$. The closure $C_1 \cdot \rho \vee Q([\mathbf{t}])$ is redundant in $N$ by assumption, so $R \cup \mathsf{irred}_R(N)^{\prec E_1} \models E_1$, but, since $E_1 \prec D$, we have $R \cup \mathsf{irred}_R(N)^{\prec D} \models E_1$. Similarly, $R \cup \mathsf{irred}_R(N)^{\prec D} \models E_2$. Since $\{E_1, E_2\} \models C \cdot \rho\tau$, we have $R \cup \mathsf{irred}_R(N)^{\prec D} \models C \cdot \rho\tau$, so the claim of the lemma holds. □

Soundness and compatibility with the standard notion of redundancy imply that $\mathcal{BS}^+$ is a sound and complete calculus, as shown by Theorem 5.4.4. Note that, to obtain the saturated set $N$, we can use any fair saturation strategy [13]. Furthermore, the decomposition rule can be applied an infinite number of times in a saturation, and it is even allowed to introduce an infinite number of definition predicates. In the latter case, we just need to ensure we use a well-founded $\mathcal{E}$-term ordering.

**Theorem 5.4.4.** *For $N_0$ a set of closures of the form $C \cdot \{\}$, let $N$ be a set of closures obtained by saturating $N_0$ under $\mathcal{BS}^+$. Then, $N_0$ is satisfiable if and only if $N$ does not contain the empty closure.*

*Proof.* The ($\Rightarrow$) direction follows immediately from Lemma 5.4.2. For the ($\Leftarrow$) direction, assume that $N$ is saturated under $\mathcal{BS}^+$. Then, by Lemma 5.4.3, $N$ is saturated under $\mathcal{BS}$ as well. Using the model generation method [14, 96], we can build a rewrite system $R$ such that $R^* \models \mathsf{irred}_R(N)$. Unlike for basic superposition without decomposition, the set of closures $N$ does not need to be well-constrained, so we cannot immediately assume that $R^*$ is a model of $N$. However, we can conclude that $R^* \models \mathsf{irred}_R(N_0)$: consider a closure $C \in N_0$ and its variable irreducible ground instance $C\tau$. If $C \in N$, then $R^*$ is obviously a model of $C\tau$. Furthermore, $C \notin N$ only if it is redundant in $N$; then, for any $\tau$, there are ground closures $D_i\tau \in \mathsf{irred}_R(N)$ such that $D_1\tau, \ldots, D_n\tau \models C\tau$. Since $R^* \models D_i\tau$ by assumption, we have $R^* \models C\tau$ as well.

Now consider a closure $C \in N_0$ and its (not necessarily variable irreducible) ground instance $C\eta$. Let $\eta'$ be a substitution obtained from $\eta$ by replacing each mapping $x \mapsto t$ with $x \mapsto \mathsf{nf}_R(t)$. Since the substitution part of $C$ is empty, $C\eta' \in \mathsf{irred}_R(N_0)$, so, because $R^* \models C\eta'$, we have $R^* \models C\eta$. Hence, $R^* \models N_0$, and by Lemma 5.4.2, there is a model of $N$. □

**Discussion.** We discuss the intuition behind the Theorem 5.4.4. Decomposition is essentially the structural transformation applied in the course of the theorem proving process. Since the formulae obtained by the structural transformation are equisatisfiable with the original formula, the application of the structural transformation does not affect the soundness of the calculus.

A potential problem might be that decomposition somehow interferes with markers of basic superposition. This does not occur because, for any rewrite system $R$, decomposing $C \cdot \rho$ into $C_1 \cdot \rho \vee Q([\mathbf{t}])$ and $C_2 \cdot \theta \vee \neg Q(\mathbf{x})$ actually decomposes any variable irreducible ground instance of the premise into corresponding variable irreducible ground instances of the conclusions. In this way, we do not lose any variable irreducible ground instance used in detecting a potential inconsistency of the closure set. Note that, for an arbitrary rewrite system $R$, closures $C_1 \cdot \rho \vee Q([\mathbf{t}])$ and $C_2 \cdot \theta \vee \neg Q(\mathbf{x})$ can have variable irreducible ground instances that do not correspond to a variable irreducible ground instance of $C \cdot \rho$. However, these variable irreducible ground instances do not cause problems, since decomposition is sound.

Another potential problem might arise if a closure $C \cdot \rho$ is derived and decomposed into $C_1 \cdot \rho \vee Q([\mathbf{t}])$ and $C_2 \cdot \theta \vee \neg Q(\mathbf{x})$ an infinite number of times. This might happen

if the ordering constraints on predicates require the fixed and the variable parts to be resolved on $Q([\mathbf{t}])$ and $\neg Q(\mathbf{x})$: obviously, the theorem proving process would be stuck in an infinite loop. This is avoided by requiring an inference to be eligible for decomposition, which makes decomposition compatible with the standard notion of redundancy. Hence, the fixed and the variable parts together make the original inference redundant, so the inference does not need to be repeated in a derivation.

**Notion of Eligibility.** We briefly discuss the way eligibility condition was defined in Definition 5.4.1. It essentially ensures that the closures obtained by decomposition of the conclusion of an inference $\xi$ are smaller than the main premise of $\xi$. Consider the example $\mathcal{BS}$ inference $\xi$, followed by decomposition (the unifier is $\sigma = \{x' \mapsto x\}$):

$$\frac{\neg A(x) \vee B(x) \vee C(y) \vee D(y) \quad A(x') \vee E(x')}{B(x) \vee E(x) \vee C(y) \vee D(y)} \quad \rightsquigarrow \quad \begin{array}{l} B(x) \vee C(y) \vee Q(x,y) \\ \neg Q(x,y) \vee E(x) \vee D(y) \end{array}$$

To determine if $\xi$ is eligible for decomposition, we have a problem: $\neg A(x)$ is the main literal on which the inference takes place, and it is not comparable with $\neg Q(x,y)$ (since $Q(x,y)$ contains an additional variable $y$). The remedy is to consider each ground substitution $\tau$ such that $\xi\tau$ satisfies the ordering constraints of $\mathcal{BS}$. For comparing terms, we shall assume an LPO as in Definition 5.3.3. Provided that $\neg A(x)$ is not selected, $\neg A(x)\sigma\tau \succ (B(x) \vee C(y))\sigma\tau$ must hold, implying that $x\sigma\tau \succ y\sigma\tau$. If we ensure that $A > Q$ in the LPO precedence, then $\neg A(x)\sigma\tau \succ \neg Q(x,y)\sigma\tau$, so the eligibility condition is satisfied.

On the contrary, assume that $\neg A(x)$ is selected. Then, $\neg A(x)\sigma\tau$ is not necessarily larger than $(B(x) \vee C(y))\sigma\tau$. Therefore, we cannot conclude that $x\sigma\tau \succ y\sigma\tau$, and that $\neg A(x)\sigma\tau \succ \neg Q(x,y)\sigma\tau$. Indeed, it is possible to take $\tau = \{x \mapsto a, y \mapsto f(a)\}$; now $A(x)\sigma\tau \prec Q(x,y)\sigma\tau$, regardless of the relation between $Q$ and $A$ in the LPO precedence. Hence, the eligibility condition is not satisfied.

By defining the eligibility condition based on the ground instances of closures, we achieve a higher level of generality. However, in most cases decomposition is applied to closures of simpler syntactic structure, for which we next derive simpler and sufficient eligibility tests.

**Proposition 5.4.5.** *Let $\xi$ be a $\mathcal{BS}$ inference as in Definition 5.4.1. If $\neg Q(\mathbf{t}) \prec L_m\eta\sigma$, then $\xi$ is eligible for superposition.*

*Proof.* Since $\prec$ is stable under substitutions, we have $\neg Q(\mathbf{t})\tau \prec L_m\eta\sigma\tau$ for each $\tau$. □

**Proposition 5.4.6.** *Let $\xi$ be a $\mathcal{BS}$ inference as in Definition 5.4.1. If the side premise $D_s \cdot \eta$ contains a literal $L \cdot \eta$ such that $\neg Q(\mathbf{t}) \prec L\eta\sigma$, then $\xi$ is eligible for superposition.*

*Proof.* For $\xi$ a $\mathcal{BS}$ inference and $\tau$ a ground substitution as in Definition 5.4.1, by the ordering conditions of $\mathcal{BS}$, we have $L_s\eta\sigma\tau \prec L_m\eta\sigma\tau$, where $L_s \cdot \eta$ is the maximal literal of the side premise. Since $\neg Q(\mathbf{t}) \prec L\eta\sigma$ by assumption, and $\prec$ is stable under substitutions, we also have that $\neg Q(\mathbf{t})\tau \prec L\eta\sigma\tau \prec L_s\eta\sigma\tau \prec L_m\eta\sigma\tau$. □

**Combining Decomposition with Other Calculi.** For any other sound clausal calculus, Lemma 5.4.2 applies identically. Furthermore, for any calculus compatible with the standard notion of redundancy [13], Lemma 5.4.3 can be proved in a similar manner with minor modifications.

### 5.4.2 Deciding $\mathcal{ALCHIQ}$ by Decomposition

In this subsection, we extend the decision procedure from Section 5.3 with the decomposition rule from Subsection 5.4.1, and thus obtain a decision procedure for checking satisfiability of $\mathcal{ALCHIQ}$ knowledge bases.

**Definition 5.4.7.** *Let $\mathcal{BS}^{+}_{DL}$ be the $\mathcal{BS}_{DL}$ calculus, where conclusions are decomposed according to the following table whenever possible, for an arbitrary term $t$:*

$$D \cdot \rho \vee R([t], [f(t)]) \rightsquigarrow \begin{array}{r} D \cdot \rho \vee Q_{R,f}([t]) \\ \neg Q_{R,f}(x) \vee R(x, [f(x)]) \end{array}$$

$$D \cdot \rho \vee R([f(x)], x) \rightsquigarrow \begin{array}{r} D \cdot \rho \vee Q_{\mathsf{Inv}(R),f}(x) \\ \neg Q_{\mathsf{Inv}(R),f}(x) \vee R([f(x)], x) \end{array}$$

*The precedence of an LPO is $f > c > P > Q_{S,f} > \mathsf{T}$, for each function symbol $f$, constant symbol $c$, nondefinition predicate $P$, and definition predicate $Q_{S,f}$.*

By Definition 5.4.1, for a (possibly inverse) role $S$ and a function symbol $f$, the definition predicate $Q_{S,f}$ is unique. Furthermore, a strict application of Definition 5.4.1 would require introducing a distinct definition predicate $Q'_{R,f}$ for $R([f(x)], x)$. However, by the operator $\pi$ for translating $KB$ into first-order logic, $R([f(x)], x)$ and $\mathsf{Inv}(R)(x, [f(x)])$ are logically equivalent. Therefore, $Q_{\mathsf{Inv}(R),f}$ can be used as the definition predicate for $R([f(x)], x)$ instead of $Q'_{R,f}$, thus avoiding the need to introduce an additional predicate in the second form of decomposition in Definition 5.4.7. This optimization is not essential for the correctness of our results; however, it is good practice to keep the number of predicate symbols minimal.

Because decomposition ensures that all non-$\mathcal{ALCHIQ}$-closures derived in a saturation are replaced with $\mathcal{ALCHIQ}$-closures, $\mathcal{BS}^{+}_{DL}$ decides satisfiability of $\mathcal{ALCHIQ}$ knowledge bases. Furthermore, since the definition predicate $Q_{R,f}$ is unique for a pair of role and function symbols $R$ and $f$, at most a polynomial number of definition predicates is introduced during saturation, so the result of Theorem 5.4.4 applies. Since the number of $\mathcal{ALCHIQ}$-closures is finite according to Lemma 5.3.9, a saturation by $\mathcal{BS}^{+}_{DL}$ terminates.

**Theorem 5.4.8.** *For an $\mathcal{ALCHIQ}$ knowledge base $KB$, saturation of $\Xi(KB)$ by $\mathcal{BS}^{+}_{DL}$ decides satisfiability of $KB$, and runs in time exponential in $|KB|$, for unary coding of numbers in input.*

*Proof.* We first show that inferences of $\mathcal{BS}^+_{DL}$, when applied to $\mathcal{ALCHIQ}$-closures, always produce $\mathcal{ALCHIQ}$-closures. The proof of Lemma 5.3.6 applies even if conditions (*iii*)–(*vi*) from Table 5.2 do not hold; the only exception is a superposition into a generator closure $\mathbf{P^f}(x) \vee R(x, f(x))$. For the latter, there are these three possibilities, depending on the structure of the premise that superposition is performed from:

- Superposition from a closure of type 5, 6, or 8 of the form $[f(t)] \approx [g(t)] \vee D \cdot \rho$, where $t$ is either a variable $x'$, a term $g(x')$ or a constant $a$, results in a closure of the form $\mathbf{P^f}([t]) \vee R([t], [g(t)]) \vee D \cdot \rho$, which is decomposed into a closure of type 3 and a closure of type 5, 6, or 8.
- Superposition from a closure of type 6 of the form $[f(g(x'))] \approx x' \vee D \cdot \rho$ results in a closure of the form $\mathbf{P^f}([g(x')]) \vee R([g(x')], x') \vee D \cdot \rho$. This closure is decomposed into a closure of type 4, and a closure of type 5 or 6. Since $R([g(x')], x')$ and $\mathsf{Inv}(R)(x', [g(x')])$ are logically equivalent due to the translation operator $\pi$, the predicate $Q_{\mathsf{Inv}(R),f}$ can be used as the definition predicate for $R([g(x')], x')$.
- Superposition from a closure of type 8 of the form $[f(a)] \approx [b] \vee D \cdot \rho$ results in a closure of the form $\mathbf{P^f}(a) \vee R([a], [b]) \vee D \cdot \rho$, which is of type 8.

Hence, decomposition ensures that the conclusion of each inference of $\mathcal{BS}^+_{DL}$ is an $\mathcal{ALCHIQ}$-closure. Note that $R(t, f(t)) = L_m\eta\sigma$ is the maximal literal of the main premise after applying the most general unifier $\sigma$ and, since $R > Q_{R,g}$ in the precedence of the LPO of $\mathcal{BS}^+_{DL}$, $\neg Q_{R,g}(t) \prec R(t, f(t))$. Hence, by Proposition 5.4.5, the first two superposition inferences from the previous list are eligible for decomposition. Also, note that decomposition can be performed after resolving a closure of type 3 or 4 with a closure of type 2 (eligibility is ensured also by Proposition 5.4.5), but this is not essential to obtain a decision procedure.

To show the claim of this theorem, let $r$ be the number of roles, and $f$ the number of function symbols occurring in $\Xi(KB)$; as in Lemma 5.3.9, both $r$ and $f$ are linear in $|KB|$ for unary coding of numbers. The number of definition predicates $Q_{R,f}$ introduced by decomposition is bounded by $r \cdot f$, which is quadratic in $|KB|$, so the number of different predicates is polynomial in $|KB|$. Hence, Lemma 5.3.9 applies in this case as well, so the maximal set of closures derived in a saturation is at most exponential in $|KB|$ for unary coding of numbers. After deriving this set, all inferences of $\mathcal{BS}^+_{DL}$ are redundant and the saturation terminates. Since $\mathcal{BS}^+_{DL}$ is a sound and complete calculus by Theorem 5.4.4, it decides satisfiability of $\Xi(KB)$ and, by Lemma 5.3.2, of $KB$ as well, in time that is exponential in $|KB|$. □

Although Theorem 5.4.8 supersedes Theorem 5.3.10, in practice it is useful to know that superposition into an $R$-generator is not necessary, provided that there is no role $S$ such that $R \sqsubseteq S$. In this way a practical implementation can be optimized not to perform inferences whose conclusions are immediately subsumed.

Note that Definition 5.4.7 applies decomposition eagerly. For example, a resolution of a closure of type 3 or 4 with a closure of type 1 or 2 produces a closure of type 3; similarly, a superposition from $[f(x)] \approx [g(x)] \vee C(x)$ into $D(x) \vee R(x, f(x))$ produces $D(x) \vee C(x) \vee R(x, [g(x)])$, which is also of type 3. By Definition 5.4.7, all these conclusions will be decomposed, even though they are $\mathcal{ALCHIQ}$-closures. In such cases, decomposition could be made optional: it may be performed, but it is not strictly necessary to obtain termination.

For most knowledge bases, not all possible predicates $Q_{R,f}$ are introduced in a saturation of $\Xi(KB)$. Rather, only predicates from the set $\mathsf{gen}(KB)$, defined in the following definition, can be introduced by decomposition. This set is used in the algorithm for reducing an $\mathcal{ALCHIQ}$ knowledge base $KB$ to a disjunctive datalog program from Chapter 7. Namely, the algorithm presented there requires all inferences among nonground clauses to be performed before any ground inferences, so, in Section 7.2, we append $\mathsf{gen}(KB)$ to $\Xi(KB)$ in advance, before saturation.

**Definition 5.4.9.** *For an $\mathcal{ALCHIQ}$ knowledge base $KB$, $\mathsf{gen}(KB)$ is the set of all closures of the form $\neg Q_{R,f}(x) \vee R(x, [f(x)])$, where $R \neq \mathsf{role}(f)$ and a role $S$ exists such that the following conditions are satisfied:*

- *$S$ occurs in an at-least number restriction under positive, or in an at-most number restriction under negative polarity in $KB$;*
- *$R \sqsubseteq^* S$ or $\mathsf{Inv}(R) \sqsubseteq^* S$;*
- *$\mathsf{role}(f) \sqsubseteq^* S$ or $\mathsf{Inv}(\mathsf{role}(f)) \sqsubseteq^* S$.*

**Lemma 5.4.10.** *In a saturation of $\Xi(KB)$ by $\mathcal{BS}^+_{DL}$, the decomposition rule introduces only definition closures from the set $\mathsf{gen}(KB)$.*

*Proof.* A definition predicate $Q_{R,f}$ is introduced in a saturation by decomposing the conclusion of superposition from a literal $[g(f(x))] \approx x$ with $\mathsf{role}(g) \neq \mathsf{Inv}(\mathsf{role}(f))$, or from a literal $[g(t)] \approx [f(t)]$ with $\mathsf{role}(g) \neq \mathsf{role}(f)$, into $\mathbf{P^g}(x) \vee R(x, g(x))$. Literals of the form $[g(f(x))] \approx x$ or $[g(t)] \approx [f(t)]$ are generated only by a hyperresolution with a closure of type 7, obtained by translating an at-least restriction on some role $S$ into clausal form. Such a hyperresolution inference is possible only if there are closures of the form $\mathbf{P_1}(x) \vee S(x, \langle f(x) \rangle)$ or $\mathbf{P_1}(x) \vee \mathsf{Inv}(S)([f(x)], x)$, and $\mathbf{P_2}(x) \vee S(x, \langle g(x) \rangle)$ or $\mathbf{P_2}(x) \vee \mathsf{Inv}(S)([g(x)], x)$. Finally, the closures of the latter form can be derived only if $R \sqsubseteq^* S$ or $\mathsf{Inv}(R) \sqsubseteq^* S$, and $\mathsf{role}(f) \sqsubseteq^* S$ or $\mathsf{Inv}(\mathsf{role}(f)) \sqsubseteq^* S$. □

### 5.4.3 Safe Role Expressions

A prominent limitation of $\mathcal{ALCHIQ}$ is a rather restricted form of axioms on roles. These limitations can be partially overcome by allowing *safe* Boolean role expressions to occur in TBox and ABox axioms. The resulting logic is called $\mathcal{ALCHIQb}$, and can be viewed as the union of $\mathcal{ALCHIQ}$ and $\mathcal{ALCIQb}$ [144]. Roughly speaking, safe role

expressions are built by combining simpler role expressions using union, disjunction, and *relativized* negation of roles: relativized statements such as

$$\forall x, y : isParentOf(x, y) \rightarrow isMotherOf(x, y) \vee isFatherOf(x, y)$$

are allowed; however, fully negated statements such as

$$\forall x, y : \neg isMotherOf(x, y) \rightarrow isFatherOf(x, y)$$

are not allowed. The safety restriction is needed for the algorithm to remain in ExpTime; namely, reasoning with unsafe role expressions is NExpTime-complete [90].

**Definition 5.4.11.** *A* role expression *is a finite expression built over the set of abstract roles using the connectives* $\sqcup$, $\sqcap$, *and* $\neg$ *in the usual way. Let* $\mathsf{safe}^+$ *and* $\mathsf{safe}^-$ *be functions defined on the set of all role expressions as follows, where* $R$ *is an abstract role and* $E_{(i)}$ *are role expressions:*

$$\begin{array}{rlrl}
\mathsf{safe}^+(R) &= true & \mathsf{safe}^-(R) &= false \\
\mathsf{safe}^+(\neg E) &= \neg\mathsf{safe}^-(E) & \mathsf{safe}^-(\neg E) &= \neg\mathsf{safe}^+(E) \\
\mathsf{safe}^+(\bigsqcup E_i) &= \bigwedge \mathsf{safe}^+(E_i) & \mathsf{safe}^-(\bigsqcup E_i) &= \bigvee \mathsf{safe}^+(E_i) \\
\mathsf{safe}^+(\bigsqcap E_i) &= \bigvee \mathsf{safe}^+(E_i) & \mathsf{safe}^-(\bigsqcap E_i) &= \bigwedge \mathsf{safe}^+(E_i)
\end{array}$$

*A role expression* $E$ *is* safe *if* $\mathsf{safe}^+(E) = true$. *The description logic* $\mathcal{ALCHIQ}b$ *is obtained from* $\mathcal{ALCHIQ}$ *by allowing concepts* $\exists E.C$, $\forall E.C$, $\geq n\, E.C$ *and* $\leq n\, E.C$, *inclusion axioms* $E \sqsubseteq F$, *and ABox axioms* $E(a, b)$, *where* $E$ *is a safe role expression, and* $F$ *is any role expression. The semantics of* $\mathcal{ALCHIQ}b$ *is obtained by extending the translation operator* $\pi$ *as specified in Table 5.3, where* $E_{(i)}$ *are role expressions.*

A similar logic $\mathcal{ALCIQ}b$ was considered in [144], where it was required that each disjunct in the disjunctive normal form of the role expression contains at least one nonnegated conjunct. These two definitions coincide; we use Definition 5.4.11 because transformation into disjunctive normal form can introduce exponential blowup, which we avoid using structural transformation.

To decide satisfiability of an $\mathcal{ALCHIQ}b$ knowledge base $KB$, we extend the preprocessing of $KB$ to transform each role expression into negation-normal form, and to introduce a new name for each nonatomic role expression. The set of closures obtained by preprocessing we also denote with $\Xi(KB)$. From [99] we know that $\Xi(KB)$ can be computed in polynomial time.

Table 5.3: Semantics of Role Expressions

$$\begin{array}{rcl}
\pi(R, X, Y) &=& R(X, Y) \\
\pi(\neg E, X, Y) &=& \neg\pi(E, X, Y) \\
\pi(\bigsqcap E_i, X, Y) &=& \bigwedge \pi(E_i, X, Y) \\
\pi(\bigsqcup E_i, X, Y) &=& \bigvee \pi(E_i, X, Y)
\end{array}$$

**Theorem 5.4.12.** *For an $\mathcal{ALCHIQ}b$ knowledge base $KB$, saturation of $\Xi(KB)$ by $\mathcal{BS}^{+}_{DL}$ decides satisfiability of $KB$ in time that is exponential in $|KB|$, for unary coding of numbers in input.*

*Proof.* In addition to $\mathcal{ALCHIQ}$ closures, $\Xi(KB)$ can contain closures obtained by translating role expressions. For a role expression $E$ occurring in concepts $\exists E.C$ and $\geq n\, E.C$ under positive polarity, or in concepts $\forall E.C$ and $\leq n\, E.C$ under negative polarity, structural transformation introduces a formula of the following form, where $Q$ is a predicate:

$$\forall x, y : Q(x, y) \rightarrow \pi(E, x, y) \tag{5.19}$$

For a role expression $E$ occurring in concepts $\exists E.C$ and $\geq n\, E.C$ under negative polarity, or in concepts $\forall E.C$ and $\leq n\, E.C$ under positive polarity, the structural transformation introduces a formula of the following form, where $Q$ is a predicate:

$$\forall x, y : \pi(E, x, y) \rightarrow Q(x, y) \tag{5.20}$$

Finally, for an inclusion axiom $E \sqsubseteq F$, the structural transformation introduces one formula of the form (5.19) and one of the form (5.20). Regardless of whether $E$ is safe in (5.19) or not, the structural transformation and clausification of (5.19) produces closures containing the literal $\neg Q(x, y)$. Furthermore, $E$ is safe in (5.20), so each disjunct in the disjunctive normal form of $E$ contains at least one positive conjunct. Hence, each disjunct in the conjunctive normal form of $\neg\pi(E, x, y)$ contains at least one literal of the form $\neg S(x, y)$, so the structural transformation and clausification of (5.20) also produce a closure with at least one such literal. Hence, all closures produced by role expressions have the form (5.21), where $n > 0$ and $m \geq 0$:

$$\neg R_1(x, y) \vee \ldots \vee \neg R_n(x, y) \vee S_1(x, y) \vee \ldots \vee S_m(x, y) \tag{5.21}$$

Such a closure always contains at least one negative literal that is selected, so the closure can participate only in hyperresolution on all negative literals. If one of the side premises is a closure of type 8 with a maximal literal $R(\langle a\rangle, \langle b\rangle)$, no side premise can have a maximal literal of the form $R(x, \langle f(x)\rangle)$ or $R([f(x)], x)$ (because $f(x)$ does not unify with a constant). Hence, the resolvent is a ground closure of type 8. Furthermore, if one side premise has a maximal literal of the form $R(x, \langle f(x)\rangle)$, then no side premise can have a maximal literal of the form $R'([g(x')], x')$, as this would require unifying $x$ with $g(x')$ and $f(x)$ with $x'$ simultaneously, which is not possible due to the occurs-check in unification [7]. If all side premises have a maximal literal of the form $R_i(x_i, \langle f(x_i)\rangle)$, the resolvent has the form (5.22), for $\mathbf{S}(s, t) = S_1(s, t) \vee \ldots \vee S_m(s, t)$:

$$\mathbf{P}(x) \vee \mathbf{S}(x, [f(x)]) \tag{5.22}$$

This closure can be decomposed into (5.23)–(5.25):

$$\mathbf{P}(x) \vee Q_{S_1,f}(x) \vee \ldots \vee Q_{S_m,f}(x) \tag{5.23}$$

$$\neg Q_{S_1,f}(x) \vee S_1(x, [f(x)]) \tag{5.24}$$

$$\vdots$$

$$\neg Q_{S_m,f}(x) \vee S_m(x, [f(x)]) \tag{5.25}$$

Finally, if all side premises have a maximal literal of the form $R_i([f(x_i)], x_i)$, the resolvent has the form (5.26):

$$\mathbf{P}(x) \vee \mathbf{S}([f(x)], x) \tag{5.26}$$

This closure can further be decomposed into (5.27)–(5.29). Since $S_i([f(x)], x)$ and $\mathsf{Inv}(S_i)(x, [f(x)])$ are logically equivalent due to the translation operator $\pi$, the predicate $Q_{\mathsf{Inv}(S_i),f}$ can be used as the definition predicate for $S_i([f(x)], x)$.

$$\mathbf{P}(x) \vee Q_{\mathsf{Inv}(S_1),f}(x) \vee \ldots \vee Q_{\mathsf{Inv}(S_m),f}(x) \tag{5.27}$$

$$\neg Q_{\mathsf{Inv}(S_1),f}(x) \vee S_1([f(x)], x) \tag{5.28}$$

$$\vdots$$

$$\neg Q_{\mathsf{Inv}(S_m),f}(x) \vee S_m([f(x)], x) \tag{5.29}$$

Hence, we ensure that the conclusions of all inferences are $\mathcal{ALCHIQ}$-closures, so Lemma 5.3.6 applies for $\mathcal{ALCHIQ}b$ as well. Furthermore, the number of literals in a closure of type (5.21) is linear in the size of the role expression, so the claim of this theorem holds in the same way as for Theorem 5.4.8. □

Note that role safety is important for Theorem 5.4.12 because it ensures that closures of type (5.21) always contain a negative literal, which is then selected. If this were not the case, a closure of the form $R(x, y)$ might participate in resolution with a closure $\mathbf{P_1}(x) \vee \neg R(x, y) \vee \mathbf{P_2}(x)$, producing a closure $\mathbf{P_1}(x) \vee \mathbf{P_2}(y)$. This closure does not match any closure from Table 5.2, which complicates the decision procedure. Since $\mathbf{P_1}(x)$ and $\mathbf{P_2}(y)$ do not have variables in common, a possible solution is to don't-know nondeterministically assume that either disjunct is true, and thus reduce the problematic closure to an $\mathcal{ALCHIQ}$ closure. This increases the complexity of the algorithm from ExpTime to NExpTime as required: in [90] it was shown that reasoning in a description logic with unsafe role expressions is NExpTime-complete. Unfortunately, the presented transformation is not applicable to all problematic closures, so we leave solving the general problem to future research.

## 5.5 Example

In Subsection 3.1.1, we introduced two knowledge bases $KB_1$ and $KB_2$, and showed, by a model-theoretic argument, that $KB_1 \cup KB_2 \models \exists hasVD.Adpt3DAcc(pc_1)$. In this section, we show that this entailment holds using a proof-theoretic argument.

Our algorithm is refutational—that is, it is capable only of detecting a contradiction. Since $\neg(\exists hasVD.Adpt3DAcc(pc_1)) \equiv \forall hasVD.\neg Adpt3DAcc(pc_1)$, we set $KB'$ to be defined as follows, and show it to be unsatisfiable:

$$KB' = KB_1 \cup KB_2 \cup \{\forall hasVD.\neg Adpt3DAcc(pc_1)\} \tag{5.30}$$

**Eliminating Transitivity Axioms.** The first step in checking satisfiability of $KB'$ is to eliminate transitivity axioms by computing $\Omega(KB')$. We start with computing the concept closure of $KB'$. For the sake of brevity, we omit concepts that can be simplified using standard identities.

$\mathsf{clos}(KB') = \{$

| | |
|---|---|
| $\forall\, hasAdpt.Adpt, Adpt,$ | (from 3.1) |
| $\forall\, has3DAcc.3DAcc, 3DAcc,$ | (from 3.2) |
| $\neg Adpt \sqcup \neg 3DAcc \sqcup Adpt3DAcc, \neg Adpt, \neg 3DAcc, Adpt3DAcc,$ | (from 3.3) |
| $\neg Adpt \sqcup VD, VD,$ | (from 3.4) |
| $\neg 3DAcc \sqcup VD,$ | (from 3.5) |
| $\neg GrWS \sqcup PC, \neg GrWS, PC,$ | (from 3.6) |
| $\neg GaPC \sqcup (GrWS \sqcap PC), \neg GaPC, GrWS \sqcap PC, GrWS,$ | (from 3.7) |
| $\neg PC \sqcup \exists\, hasAdpt.\top, \neg PC, \exists\, hasAdpt.\top,$ | (from 3.12) |
| $\neg GrWS \sqcup \exists\, has3DAcc.\top, \exists\, has3DAcc.\top,$ | (from 3.13) |
| $\neg GaPC \sqcup \leq 1\, hasVD.VD, \leq 1\, hasVD.VD, \neg VD,$ | (from 3.14) |
| $\neg PCI \sqcup \forall\, contains.PCI, \neg PCI, \forall\, contains.PCI, PCI,$ | (from 3.15) |
| $GaPC,$ | (from 3.16) |
| $\forall hasVD.\neg Adpt3DAcc, \neg Adpt3DAcc$ | (from 5.30) |

$\}$

We next select all concepts of the form $\forall R.C$ from $\mathsf{clos}(KB')$, where $R$ is transitive or has a transitive subrole; in our example, the only such concept is $\forall\, contains.PCI$. Finally, to construct $\Omega(KB')$, we remove from $KB'$ the transitivity axiom (3.11) and add the axiom (5.31):

$$\forall\, contains.PCI \sqsubseteq \forall\, contains.\forall\, contains.PCI \tag{5.31}$$

**Translation into Closures.** To check satisfiability of $\Omega(KB')$ using basic superposition, the knowledge base must be extensionally reduced; that is, ABox axioms should only contain literal concepts. Hence, we replace (5.30) by (5.32) and (5.33):

$$Q_1 \sqsubseteq \forall hasVD.\neg Adpt3DAcc \tag{5.32}$$

$$Q_1(pc_1) \tag{5.33}$$

We now apply the clausification algorithm from Definition 5.3.1. The following closures correspond to TBox and RBox axioms of $\Omega(KB')$ other than (5.31):

$$\neg hasAdpt(x,y) \vee Adpt(y) \quad (3.1)$$ (5.34)
$$\neg has3DAcc(x,y) \vee 3DAcc(y) \quad (3.2)$$ (5.35)
$$\neg Adpt(x) \vee \neg 3DAcc(x) \vee Adpt3DAcc(x) \quad (3.3)$$ (5.36)
$$\neg Adpt(x) \vee VD(x) \quad (3.4)$$ (5.37)
$$\neg 3DAcc(x) \vee VD(x) \quad (3.5)$$ (5.38)
$$\neg GrWS(x) \vee PC(x) \quad (3.6)$$ (5.39)
$$\neg GaPC(x) \vee GrWS(x) \quad (3.7)$$ (5.40)
$$\neg GaPC(x) \vee PC(x) \quad (3.7)$$ (5.41)
$$\neg hasAdpt(x,y) \vee hasVD(x,y) \quad (3.8)$$ (5.42)
$$\neg has3DAcc(x,y) \vee hasVD(x,y) \quad (3.9)$$ (5.43)
$$\neg hasVD(x,y) \vee contains(x,y) \quad (3.10)$$ (5.44)
$$\neg PC(x) \vee hasAdpt(x, f(x)) \quad (3.12)$$ (5.45)
$$\neg GrWS(x) \vee has3DAcc(x, g(x)) \quad (3.13)$$ (5.46)
$$\neg GaPC(x) \vee \neg hasVD(x,y_1) \vee \neg hasVD(x,y_2) \vee y_1 \approx y_2 \vee \neg VD(y_1) \vee \neg VD(y_2) \quad (3.14)$$ (5.47)
$$\neg PCI(x) \vee \neg contains(x,y) \vee PCI(y) \quad (3.15)$$ (5.48)
$$\neg Q_1(x) \vee \neg hasVD(x,y) \vee \neg Adpt3DAcc(y) \quad (5.32)$$ (5.49)

Axiom (5.31) contains nonliteral subconcepts, so it must be transformed using structural transformation. Hence, we compute $\mathsf{Def}(C)$ for $C$ as follows:

$$C = \neg(\forall\, contains.PCI) \sqcup \forall\, contains.\forall\, contains.PCI$$

We start by setting $\Lambda(C) = \{1.1, 2.2\}$. Namely, $C|_{1.1} = \forall\, contains.PCI$, and each subconcept of $C|_{1.1}$ is atomic; the situation is similar for $C|_{2.2} = \forall contains.PCI$. Next, we introduce new names $Q_2$ and $Q_3$ for $C|_{1.1}$ and $C|_{2.2}$, respectively. Furthermore, $\mathsf{pol}(C, 1.1) = -1$, but $\mathsf{pol}(C, 2.2) = 1$, so, after skipping some intermediate steps, we obtain the following set $\mathsf{Def}(C)$:

$$\mathsf{Def}(C) = \{Q_2 \sqcup \neg(\forall\, contains.PCI)\} \cup \{\neg Q_3 \sqcup \forall\, contains.PCI\} \cup \mathsf{Def}(\neg Q_2 \sqcup \forall\, contains.Q_3)$$

We now recursively compute $\mathsf{Def}(D)$ for $D = \neg Q_2 \sqcup \forall\, contains.Q_3$. The subconcepts of $D$ are all literal except $\forall\, contains.Q_3$, but this is the only nonatomic subconcept in a disjunction, so renaming it is not strictly necessary; thus, $\mathsf{Def}(D) = \{D\}$. By clausifying $\mathsf{Def}(C)$, we obtain the following closures:

$$Q_2(x) \vee contains(x, h(x)) \quad (5.31)$$ (5.50)
$$Q_2(x) \vee \neg PCI(h(x)) \quad (5.31)$$ (5.51)
$$\neg Q_3(x) \vee \neg contains(x,y) \vee PCI(y) \quad (5.31)$$ (5.52)
$$\neg Q_2(x) \vee \neg contains(x,y) \vee Q_3(y) \quad (5.31)$$ (5.53)

Finally, we append the following ABox closures:

$$GaPC(pc_1) \quad (3.16)$$ (5.54)
$$Q_1(pc_1) \quad (5.33)$$ (5.55)

**Term Ordering.** To obtain the lexicographic ordering for $\mathcal{BS}^+_{DL}$, we use the precedence $>$ where all function symbols are larger than all constants, all constants are larger than all predicate symbols, and symbols of equal type are compared alphabetically. An exception are definition predicates starting with the letter $Q$, which, because of Definition 5.4.7, must be smallest. Furthermore, we select all negative binary literals. The $\mathcal{E}$-terms that participate in inferences because they are either selected or maximal in the $\mathcal{E}$-term ordering are displayed $\boxed{\text{like this}}$.

**Saturation by $\mathcal{BS}^+_{DL}$.** We now saturate the set of closures by basic superposition. Since this set is fairly large, we need a strategy in applying $\mathcal{BS}^+_{DL}$ inferences that ensures we do not miss an inference. To that purpose, we use the *DISCOUNT loop* algorithm [133]. The algorithm is presented formally in Subsection 12.3.1; here, we give a brief overview in order to make this section self-contained. The algorithm maintains two sets of closures: the set $U$ of unprocessed closures, and the set $W$ of worked-off closures. Initially, $U = \Xi(\Omega(KB'))$, and $W = \emptyset$. In each iteration of the algorithm, a *given closure* $C$ is (don't-care nondeterministically) selected in $U$. Then, all possible $\mathcal{BS}^+_{DL}$ inferences with $C$ and premises from $W$ are computed; we denote the set of consequences with $S$. Next, $C$ is removed from $U$ and added to $W$. Finally, all closures from $S$ that are nonredundant w.r.t. $W$ are added to $U$. The algorithm terminates when $U$ becomes empty. Note that the algorithm has an important property: after each iteration, all possible inferences among closures in $W$ have been performed. Therefore, when the algorithm terminates, $W$ is the saturated set of closures.

An iteration of the algorithm is presented using the following notation. $C$ is the given closure, and is numbered in the set $W$ with (xx.xx). Closures $S_1, \ldots, S_n$ are the conclusions of all inferences involving $C$ and closures from $W$. A label R(yy.yy) means that $S_1$ is derived by resolving $C$ with the closure (yy.yy) from $W$; a label S(zz.zz) means that $S_n$ is derived by superposition of $C$ and the closure (zz.zz) from $W$. Thus, at any given instant, the set $W$ consists of closures marked with $\Rightarrow$. At start, the set $U$ consists of closures (5.34)–(5.55); as the algorithm proceeds, it also contains closures derived in a previous iteration, but not yet moved to $W$.

$\Rightarrow$ (xx.xx) $C$

---

R(yy.yy) $S_1$

$\vdots$

S(zz.zz) $S_n$

We now show the inferences of $\mathcal{BS}^+_{DL}$ on $\Xi(\Omega(KB'))$. To make the presentation shorter, we perform only decomposition inferences that are strictly necessary.

$\Rightarrow$ (5.56) $\neg PC(x) \vee \boxed{hasAdpt(x, f(x))}$

$\Rightarrow$ (5.57) $\neg GrWS(x) \vee \boxed{has3DAcc(x, g(x))}$

$\Rightarrow$ (5.58) $\boxed{\neg hasAdpt(x,y)} \vee hasVD(x,y)$

R(5.56) $\neg PC(x) \vee hasVD(x,[f(x)])$

$\Rightarrow$ (5.59) $\boxed{\neg has3DAcc(x,y)} \vee hasVD(x,y)$

R(5.57) $\neg GrWS(x) \vee hasVD(x,[g(x)])$

$\Rightarrow$ (5.60) $\neg PC(x) \vee \boxed{hasVD(x,[f(x)])}$

$\Rightarrow$ (5.61) $\neg GrWS(x) \vee \boxed{hasVD(x,[g(x)])}$

$\Rightarrow$ (5.62) $\boxed{\neg hasAdpt(x,y)} \vee Adpt(y)$

R(5.56) $\neg PC(x) \vee Adpt([f(x)])$

$\Rightarrow$ (5.63) $\boxed{\neg has3DAcc(x,y)} \vee 3DAcc(y)$

R(5.57) $\neg GrWS(x) \vee 3DAcc([g(x)])$

$\Rightarrow$ (5.64) $\neg PC(x) \vee \boxed{Adpt([f(x)])}$

$\Rightarrow$ (5.65) $\neg GrWS(x) \vee \boxed{3DAcc([g(x)])}$

$\Rightarrow$ (5.66) $\neg Adpt(x) \vee \boxed{VD(x)}$

$\Rightarrow$ (5.67) $\neg 3DAcc(x) \vee \boxed{VD(x)}$

$\Rightarrow$ (5.68) $\neg GaPC(x) \vee \boxed{\neg hasVD(x,y_1)} \vee \boxed{\neg hasVD(x,y_2)} \vee y_1 \approx y_2 \neg VD(y_1) \vee \neg VD(y_2)$

R(5.60;5.61) $\neg GaPC(x) \vee \neg PC(x) \vee \neg GrWS(x) \vee [g(x)] \approx [f(x)] \vee \neg VD([g(x)]) \vee \neg VD([f(x)])$

$\Rightarrow$ (5.69) $\neg GaPC(x) \vee \neg PC(x) \vee \neg GrWS(x) \vee [g(x)] \approx [f(x)] \vee \boxed{\neg VD([g(x)])} \vee \neg VD([f(x)])$

R(5.66) $\neg GaPC(x) \vee \neg PC(x) \vee \neg GrWS(x) \vee [g(x)] \approx [f(x)] \vee \neg Adpt([g(x)]) \vee \neg VD([f(x)])$

R(5.67) $\neg GaPC(x) \vee \neg PC(x) \vee \neg GrWS(x) \vee [g(x)] \approx [f(x)] \vee \neg 3DAcc([g(x)]) \vee \neg VD([f(x)])$

$\Rightarrow$ (5.70) $\neg GaPC(x) \vee \neg PC(x) \vee \neg GrWS(x) \vee [g(x)] \approx [f(x)] \vee \boxed{\neg 3DAcc([g(x)])} \vee \neg VD([f(x)])$

R(5.65) $\neg GaPC(x) \vee \neg PC(x) \vee \neg GrWS(x) \vee [g(x)] \approx [f(x)] \vee \neg VD([f(x)])$

$\Rightarrow$ (5.71) $\neg GaPC(x) \vee \neg PC(x) \vee \neg GrWS(x) \vee \boxed{[g(x)]} \approx [f(x)] \vee \neg VD([f(x)])$

S(5.57) $\neg GaPC(x) \vee \neg PC(x) \vee \neg GrWS(x) \vee has3DAcc(x,[f(x)]) \vee \neg VD([f(x)])$

The conclusion of the last inference is not an $\mathcal{ALCHIQ}$-closure, so we must decompose it. We introduce a new predicate $Q_4$, replace the conclusion with (5.72) and (5.73), and continue the saturation.

$\Rightarrow$ (5.72) $\neg Q_4(x) \vee \boxed{has3DAcc(x,[f(x)])}$

R(5.63) $\neg Q_4(x) \vee 3DAcc([f(x)])$

R(5.59) $\neg Q_4(x) \vee hasVD(x,[f(x)])$

$\Rightarrow$ (5.73) $\neg GaPC(x) \vee \neg PC(x) \vee \neg GrWS(x) \vee Q_4(x) \vee \boxed{\neg VD([f(x)])}$

R(5.66) $\neg GaPC(x) \vee \neg PC(x) \vee \neg GrWS(x) \vee Q_4(x) \vee \neg Adpt([f(x)])$
R(5.67) $\neg GaPC(x) \vee \neg PC(x) \vee \neg GrWS(x) \vee Q_4(x) \vee \neg 3DAcc([f(x)])$

$\Rightarrow$ (5.74) $\neg GaPC(x) \vee \neg PC(x) \vee \neg GrWS(x) \vee Q_4(x) \vee \boxed{\neg Adpt([f(x)])}$

R(5.64) $\neg GaPC(x) \vee \neg PC(x) \vee \neg GrWS(x) \vee Q_4(x)$

$\Rightarrow$ (5.75) $\neg Q_4(x) \vee \boxed{3DAcc([f(x)])}$

$\Rightarrow$ (5.76) $\neg Q_4(x) \vee \boxed{hasVD(x, [f(x)])}$

R(5.68;5.61) $\neg GaPC(x) \vee \neg Q_4(x) \vee \neg PC(x) \vee [g(x)] \approx [f(x)] \neg VD([g(x)]) \vee \neg VD([f(x)])$

$\Rightarrow$ (5.77) $\neg Q_1(x) \vee \boxed{\neg hasVD(x, y)} \vee \neg Adpt3DAcc(y)$

R(5.60) $\neg Q_1(x) \vee \neg PC(x) \vee \neg Adpt3DAcc([f(x)])$
R(5.61) $\neg Q_1(x) \vee \neg GrWS(x) \vee \neg Adpt3DAcc([g(x)])$
R(5.76) $\neg Q_1(x) \vee \neg Q_4(x) \vee \neg Adpt3DAcc([f(x)])$

$\Rightarrow$ (5.78) $\neg Q_1(x) \vee \neg Q_4(x) \vee \boxed{\neg Adpt3DAcc([f(x)])}$

$\Rightarrow$ (5.79) $\neg Adpt(x) \vee \neg 3DAcc(x) \vee \boxed{Adpt3DAcc(x)}$

R(5.78) $\neg Adpt([f(x)]) \vee \neg 3DAcc([f(x)]) \vee \neg Q_1(x) \vee \neg Q_4(x)$

$\Rightarrow$ (5.80) $\boxed{\neg Adpt([f(x)])} \vee \neg 3DAcc([f(x)]) \vee \neg Q_1(x) \vee \neg Q_4(x)$

R(5.64) $\neg PC(x) \vee \neg 3DAcc([f(x)]) \vee \neg Q_1(x) \vee \neg Q_4(x)$

$\Rightarrow$ (5.81) $\neg PC(x) \vee \boxed{\neg 3DAcc([f(x)])} \vee \neg Q_1(x) \vee \neg Q_4(x)$

R(5.75) $\neg PC(x) \vee \neg Q_1(x) \vee \neg Q_4(x)$

$\Rightarrow$ (5.82) $\boxed{\neg PC(x)} \vee \neg Q_1(x) \vee \neg Q_4(x)$

$\Rightarrow$ (5.83) $\neg GaPC(x) \vee \boxed{\neg PC(x)} \vee \neg GrWS(x) \vee Q_4(x)$

$\Rightarrow$ (5.84) $\neg GaPC(x) \vee \boxed{PC(x)}$

R(5.82) $\neg GaPC(x) \vee \neg Q_1(x) \vee \neg Q_4(x)$
R(5.83) $\neg GaPC(x) \vee \neg GrWS(x) \vee Q_4(x)$

$\Rightarrow$ (5.85) $\neg GaPC(x) \vee \boxed{\neg GrWS(x)} \vee Q_4(x)$

$\Rightarrow$ (5.86) $\neg GaPC(x) \vee \boxed{GrWS(x)}$

R(5.85) $\neg GaPC(x) \vee \vee Q_4(x)$

$\Rightarrow$ (5.87) $\boxed{\neg GaPC(x)} \vee Q_4(x)$

$\Rightarrow$ (5.88) $\boxed{\neg GaPC(x)} \vee \neg Q_1(x) \vee \neg Q_4(x)$

$\Rightarrow$ (5.89) $\boxed{GaPC(pc_1)}$

R(5.87) $Q_4([pc_1])$
R(5.88) $\neg Q_1([pc_1]) \vee \neg Q_4([pc_1])$

$\Rightarrow$ (5.90) $\boxed{Q_4([pc_1])}$

$\Rightarrow$ (5.91) $\neg Q_1([pc_1])\boxed{\vee \neg Q_4([pc_1])}$

R(5.90) $\neg Q_1([pc_1])$

$\Rightarrow$ (5.92) $\boxed{\neg Q_1([pc_1])}$

$\Rightarrow$ (5.93) $\boxed{Q_1(pc_1)}$

R(5.92) $\square$

$\Rightarrow$ (5.94) $\square$

In step (5.94), we added the empty closure to $W$. This makes all other closures redundant, so the saturation terminates. Moreover, $\Xi(\Omega(KB'))$ is unsatisfiable, and so is $KB'$, so $KB_1 \cup KB_2 \models \exists hasVD.Adpt3DAcc(pc_1)$.

## 5.6 Related Work

Decision procedures for various logics were investigated in the field of automated theorem proving from its early days. Three such procedures were already implemented by Wang in 1960: a procedure capable of deciding validity in propositional logic, a procedure for deriving theorems in propositional logic, and a procedure for deciding validity in the so-called **AE**-fragment of first-order logic, consisting of first-order formulae with a quantifier prefix of the form $\forall x_1 \ldots \forall x_m \exists y_1 \ldots \exists y_n$.

At the beginning of the sixties, Robinson introduced the resolution principle [121] for first-order logic. Due to its simplicity, resolution soon gained popularity. Namely, a resolution-based theorem prover must implement only one inference rule, thus eliminating the complexity involved in choosing the rule to apply next. Soon after the initial work by Robinson, various refinements of resolution were developed, such as hyperresolution [122], ordered resolution [115], paramodulation [120], or lock resolution [23], to name just a few. The common goal of all of these refinements is to reduce the number of consequences generated in the theorem proving process without losing completeness. A good overview of resolution and related refinements is given in the classical textbook by Chang and Lee [26].

Soon after the introduction of the resolution principle and its refinements, attempts were made to use them to obtain efficient decision procedures for various classes of first-order logic. The first such procedure was presented by Kallick [81] for the class of formulae with the quantification prefix $\forall x_1 \forall x_2 \exists y$. This decision procedure is based

on a refinement of resolution that is incomplete for first-order logic, and is therefore difficult to extend.

In [80], Joyner has established the basic principles for deriving resolution-based decision procedures. He observed that, if clauses derivable in a saturation by a resolution refinement have a bounded term depth and clause length, then saturation necessarily terminates. By choosing appropriate refinements, he presented decision procedures for the Ackermann class (where the formulae are restricted to the quantification prefix $\exists^*\forall\exists^*$), the Monadic class (where only unary predicates are allowed), and the Maslov class (where formulae are restricted to quantification prefix $\exists^*\forall^*\exists^*$, and the formula under the quantifiers is a conjunction of binary disjunctions).

In the years to follow, the approach by Joyner was applied to numerous other decidable classes, such as the $\mathbf{E}^+$ class [142], the $\mathcal{PVD}$ class [83], and the $\mathcal{PVD}^g_=$ class [107], to name just a few. An overview of these results is given in [45].

Decidability of description logics in the resolution framework has been studied extensively in [98, 77, 76]. There, the description logic $\mathcal{ALB}$ is embedded in the DL* clausal class, which is then decided using the resolution framework by Bachmair and Ganzinger [13]. The main advantage of using this framework lies in its effective redundancy elimination methods, which were shown to be essential for the practical applicability of resolution calculi. $\mathcal{ALB}$ is a very expressive logic and allows for unsafe role expressions, but does not provide for counting quantifiers.

In [48], a decision procedure for the modal logic with a single transitive modality K4 was presented. To deal with transitivity, the algorithm is based on the ordered chaining calculus [11]. This calculus consists of inference rules aimed at optimizing theorem proving with chains of binary roles. Unfortunately, our attempts to decide $\mathcal{SHIQ}$ using ordered chaining proved unsuccessful, mainly due to certain negative chaining inferences that produced undesirable equality literals. Therefore, we adopted the approach for eliminating transitivity presented in Section 5.2.

The guarded fragment was introduced in [3] to explain and generalize the good properties of modal and description logic, such as decidability. A decision procedure by resolution with a nonliftable ordering was presented in [36]; it was later modified to handle the (loosely) guarded fragment with equality in [47] by basing the algorithm on superposition [9]. Since the basic description logic $\mathcal{ALC}$ is actually a syntactic variant of the multi-modal logic $\mathrm{K}_m$ [127], it can be embedded into the guarded fragment and decided by [47]. Using the approach from [129], certain extensions of $\mathcal{ALC}$, such as role transitivity, can be encoded into $\mathcal{ALC}$ knowledge bases, so the algorithm from [47] can decide these extensions as well. However, $\mathcal{SHIQ}$ is not a fragment of the (loosely) guarded fragment because of the counting quantifiers: equality is available in the guarded fragment, but each two pairs of free variables of a guarded formula must occur in a guard atom. In fact, in [65] it was shown that the guarded fragment has the finite-model property, which is known not to hold for $\mathcal{SHIQ}$ [4, Chapter 2], thus suggesting that other mechanisms are necessary for handling the latter logic.

$\mathcal{SHIQ}$ can easily be embedded into the two-variable fragment of first-order logic with counting quantifiers $\mathcal{C}^2$. This fragment was shown to be decidable in [54, 100], and

a decision procedure based on a combination of resolution and integer programming was presented in [110]. However, deciding satisfiability of $\mathcal{C}^2$ is NExpTime-complete [100], and $\mathcal{SHIQ}$ is an ExpTime-complete logic [144]. Thus, the decision procedure from [110] introduces an unnecessary overhead for $\mathcal{SHIQ}$. Furthermore, we do not see how to use this procure to derive the desired reduction to disjunctive datalog.

The decomposition rule is closely related to the structural transformation [108, 99, 8]. However, structural transformation is usually applied as a preprocessing step and not in the theorem proving process. In [37] and [118], *splitting by propositional symbols* was proposed that allows splitting variable-disjoint subsets of a clause and connect them by a propositional symbol. Finally, a *separation* rule was used to decide fluted logic in [128]. It was shown that resolution remains complete if the separation rule is applied a finite number of times during saturation. In contrast to these related approaches, our rule can decompose complex terms. Moreover, we demonstrate compatibility of the decomposition rule with the standard redundancy notion. Finally, extending basic superposition with decomposition is not trivial, due to the nonstandard approach to lifting employed by basic superposition.

# Chapter 6

# Reasoning with a Concrete Domain

As argued in [5], representing concrete data is essential for practical knowledge representation systems. However, existing algorithms for reasoning with concrete domains from [5, 70, 62, 87] mainly operate in tableau and automata frameworks, which nondeterministically check satisfiability of conjunctions of ground atoms. On the contrary, the algorithm from Chapter 5 is based on resolution, and it works with conjunctions of nonground disjunctions. Hence, extending the results from Chapter 5 with concrete domain reasoning is not trivial. In order to support the description logic $\mathcal{SHIQ}(\mathbf{D})$, in this chapter we present two new results.

First, in Section 6.1 we present a general approach for combining concrete domain reasoning with clausal calculi. The approach consists of two steps. A $\mathsf{c}$-*factoring* preprocessing transformation is applied first to bring a clause set into a suitable form. Next, we introduce a *concrete domain resolution* inference rule, for which we show soundness and completeness when combined with a clausal calculus whose completeness proof is based on the model generation method. Note that this is the standard technique for proving completeness of many state-of-the-art calculi, such as ordered resolution [13], basic superposition [14], or ordered chaining [11], so concrete domain resolution is applicable to a wide range of calculi.

Second, in Section 6.2 we apply these techniques to the algorithm from Chapter 5, and thus obtain a procedure for deciding satisfiability of $\mathcal{SHIQ}(\mathbf{D})$ knowledge bases. Assuming a bound on the arity of concrete predicates and an exponential procedure for checking $\mathbf{D}$-satisfiability of conjunctions of concrete predicates, extending the logic does not increase the complexity of reasoning—that is, it remains in ExpTime.

Results from this chapter were published in [150]. Due to space limitations, we did not consider there the distinction between $\mathbf{D}$- and $\mathbf{d}$-satisfiability, as discussed in Subsection 6.1.2 (for $\mathcal{SHIQ}(\mathbf{D})$ without ABox assertions of the form $\neg T(a, b^{\mathsf{c}})$, these two notions coincide).

## 6.1 Resolution with a Concrete Domain

In this section we develop a general algorithm for reasoning in first-order logic extended with an admissible concrete domain.

### 6.1.1 Preliminaries

The key step in computing the set of clauses $\mathsf{Cls}(\varphi)$ from a first-order formula $\varphi$ is to skolemize existential quantifiers, as explained in Section 2.1. Similarly, to check **D**-satisfiability of $\varphi$, skolemization can be applied as well, since, as shown by the following lemma, it does not affect **D**-satisfiability:

**Lemma 6.1.1.** *A formula $\varphi$ is* **D***-satisfiable if and only if $\mathsf{sk}(\varphi)$ is* **D***-satisfiable.*

*Proof.* The proof is identical to the case of ordinary satisfiability presented in [46]: given a **D**-model $I$ of $\varphi$, one can build a **D**-model $I'$ of $\mathsf{sk}(\varphi)$ by extending $I$ with the appropriate interpretations of new general function symbols. Conversely, new general function symbols ensure existence of existentially implied individuals. □

Hence, to check **D**-satisfiability of a formula $\varphi$, we first compute a **D**-equisatisfiable set of clauses $N = \mathsf{Cls}(\varphi)$. Furthermore, since the concrete domain **D** is admissible, we replace each literal $\neg d(\mathbf{t})$ with $\overline{d}(\mathbf{t})$. Thus, we assume without loss of generality that concrete domain predicates occur only in positive literals.

Checking satisfiability of a set of clauses $N$ in an arbitrary model is problematic, because one should consider arbitrary domains. In the case of first-order logic without concrete domains, this can conveniently be avoided by considering only Herbrand models. For the case of first-order logic extended with a concrete domain, we introduce the notion of Herbrand **D**-interpretations, as follows:

**Definition 6.1.2.** *For a function $\delta : HU_{\mathsf{c}} \to \triangle_{\mathbf{D}}$ and a ground formula $\varphi$, $\delta(\varphi)$ is the ground formula obtained by replacing each term $t^{\mathsf{c}}$ with $\delta(t^{\mathsf{c}})$. The extension of $\delta$ to sets of formulae is defined by applying $\delta$ to each set member.*

*A* Herbrand **D**-interpretation *over a signature $\Sigma$ is a pair $(I, \delta)$, where $I$ is a classical Herbrand interpretation over $\Sigma$ and $\delta : HU_{\mathsf{c}} \to \triangle_{\mathbf{D}}$ is an assignment of concrete individuals to concrete terms of $HU_{\mathsf{c}}$. A Herbrand* **D***-interpretation is* well-formed *if the following two conditions are true:*

- *$s_i \approx t_i \in I$ for $1 \leq i \leq n$ imply $\delta(f^{\mathsf{c}}(s_1, \ldots s_n)) = \delta(f^{\mathsf{c}}(t_1, \ldots t_n))$, for each general function symbol $f^{\mathsf{c}}$;*
- *$\delta(\mathbf{t}) \in d^{\mathbf{D}}$ for each concrete literal $d(\mathbf{t}) \in I$.*

*A ground clause $C^G$ is* **D**-satisfied *in a Herbrand* **D***-interpretation $(I, \delta)$, written $(I, \delta) \models_{\mathbf{D}} C^G$, if $(I, \delta)$ is well-formed and $\delta(C^G)$ is true in $\delta(I)$. A nonground clause $C$ is* **D***-satisfied in $(I, \delta)$ if all its ground instances are* **D***-satisfied in $(I, \delta)$. The notions of* **D***-models,* **D***-satisfiability and so on extend to Herbrand* **D***-interpretations as usual.*

There is a slight problem with Herbrand **D**-models as defined in the previous definition. Namely, Definition 6.1.2 does not ensure that the number of constants in the Herbrand universe matches the number of objects in $\triangle_{\mathbf{D}}$. Consider the set of clauses $N = \{=_1(x)\}$ with the concrete domain **D** such that $\triangle_{\mathbf{D}} = \{1, 2\}$. Obviously, $N$ is **D**-unsatisfiable (since 2 is not equal to 1); however, the Herbrand **D**-interpretation defined by $I = \{=_1(a)\}$ and $\delta(a) = 1$ is a Herbrand **D**-model of $N$. This discrepancy arises because $I$ does not contain a distinct constant for each element of $\triangle_{\mathbf{D}}$. To ensure that considering only Herbrand models does not affect satisfiability, we append to $N$ the (possibly infinite) set of clauses $\aleph^{\mathbf{D}} = \{=_\alpha (a_\alpha) \mid \text{ for each } \alpha \in \triangle_{\mathbf{D}}\}$, where $a_\alpha$ are new constants (note that the notion of admissibility from Definition 3.2.1 ensures that the predicates $=_\alpha$ exist). A set of clauses $N$ that contains $\aleph^{\mathbf{D}}$ is said to be **D**-*extended*.

We now show that, for **D**-extended clause sets, we can consider **D**-satisfiability only in Herbrand **D**-models.

**Lemma 6.1.3.** *A set of* **D***-extended clauses $N$ is* **D***-satisfiable if and only if a well-formed Herbrand* **D***-model of $N$ exists.*

*Proof.* ($\Leftarrow$) Let $(I, \delta)$ be a Herbrand **D**-model of $N$. We construct a model $I'$ by setting the domain $\mathcal{D}_r = HU_r$ for each sort $r \neq \mathsf{c}$, $\mathcal{D}_\mathsf{c} = \triangle_{\mathbf{D}}$, $t^{I'} = t$ for each term $t$ of sort other than $\mathsf{c}$, $(f^\mathsf{c})^{I'}(t_1, \ldots, t_n) = \delta(f^\mathsf{c}(t_1, \ldots, t_n))$, and $(t_1^{I'}, \ldots, t_n^{I'}) \in P^{I'}$ if and only if $P(t_1, \ldots, t_n) \in I$ for a predicate $P$. Because $N$ is **D**-extended, each object from $\triangle_{\mathbf{D}}$ is represented as a constant in $HU_\mathsf{c}$, so $I'$ is a **D**-model of $N$.

($\Rightarrow$) Let $N$ be **D**-satisfiable in a **D**-model $I$. Let $I'$ be an interpretation containing those ground atoms $A$ from the Herbrand base of $N$ such that $A^I$ are true in $I$. Furthermore, for each ground concrete term $t^\mathsf{c} \in HU_\mathsf{c}$, let $\delta(t^\mathsf{c}) = (t^\mathsf{c})^I$. In the same way as for first-order logic without concrete domains in [46], $I'$ is a classical Herbrand model of $N$, and $(I', \delta)$ is a **D**-model of $N$. □

### 6.1.2 d-Satisfiability

The task of finding a Herbrand **D**-model $(I, \delta)$ of a set of clauses $N$ essentially consists of two subtasks. The first one is to ensure that a Herbrand model $I$ exists; this can be done using resolution in the standard way. The second one is to ensure that an appropriate assignment $\delta$ exists. To solve the latter problem, we first consider the simpler task of finding $\delta$ that only satisfies the concrete predicates.

**Definition 6.1.4.** *A ground clause $C^G$ is* **d***-satisfied in a Herbrand* **D***-interpretation $(I, \delta)$, written $(I, \delta) \models_{\mathbf{d}} C^G$, if and only if $(I, \delta)$ is well-formed and $C^G$ is classically satisfied in $I$ (that is, it is satisfied without taking special care of concrete individuals). A nonground clause $C$ is* **d***-satisfied in $(I, \delta)$ if all ground instances of $C$ are* **d***-satisfied in $(I, \delta)$. Other notions are defined analogously.*

Note that **d**-satisfiability is a strictly weaker notion than **D**-satisfiability: if $N$ is **D**-satisfiable, it is obviously **d**-satisfiable, but the converse does not hold. For example, consider a set of clauses $N = \{R(a, b^\mathsf{c}), \neg R(a, c^\mathsf{c}), d(b^\mathsf{c}, c^\mathsf{c})\}$ and a concrete

domain $\mathbf{D}$ such that $d^{\mathbf{D}} = \{(1,1)\}$. Obviously, $I = \{R(a, b^{\mathsf{c}}), d(b^{\mathsf{c}}, c^{\mathsf{c}})\}$ is a classic Herbrand model of $N$; furthermore, for a function $\delta$ such that $\delta(b^{\mathsf{c}}) = \delta(c^{\mathsf{c}}) = 1$, we have $(\delta(b^{\mathsf{c}}), \delta(c^{\mathsf{c}})) \in d^{\mathbf{D}}$, so $N$ is **d**-satisfiable. However, $N$ is not **D**-satisfiable: $\delta(I) = \{R(a,1), d(1,1)\}$, but then $\delta(R(a, c^{\mathsf{c}}))$ is true in $\delta(I)$, so the clause $\delta(\neg R(a, c^{\mathsf{c}}))$ is not true in $\delta(I)$. Intuitively, **d**-satisfiability ensures consistency of literals containing concrete domain predicates, but not of literals with ordinary predicates containing both concrete and abstract terms. The relationship between **d**- and **D**-satisfiability can be captured using c-factors, introduced below. Please keep in mind that, as discussed in Subsection 3.2.1, $s^{\mathsf{c}} \not\approx_{\mathbf{D}} t^{\mathsf{c}}$ is a *positive* literal with a concrete predicate $\not\approx_{\mathbf{D}}$.

**Definition 6.1.5.** *Let $C$ be a clause containing a literal $\neg A$. The* c-factor *of $\neg A$ w.r.t. $C$ is the disjunction*

$$\neg A[x_1^{\mathsf{c}}]_{p_1} \ldots [x_n^{\mathsf{c}}]_{p_n} \vee x_1^{\mathsf{c}} \not\approx_{\mathbf{D}} A|_{p_1} \vee \ldots \vee x_n^{\mathsf{c}} \not\approx_{\mathbf{D}} A|_{p_n}$$

*where $x_i^{\mathsf{c}}$ are globally new variables, and $p_1, \ldots, p_n$ are exactly those positions in $A$ such that $\mathsf{sort}(A|_{p_i}) = \mathsf{c}$ and at least one condition from the following list holds:*

- *$A|_{p_i}$ is not a variable;*
- *$A|_{p_i}$ is a variable occurring in some negative literal in $C \setminus \{\neg A\}$;*
- *$A|_{p_i}$ is a variable occurring in $\neg A$ at some position other than $p_i$.*

*A clause $C'$ is a* c*-factor of $C$ if no literal from $C'$ has a* c*-factor w.r.t. $C'$, and if a sequence of clauses $C = C_0, \ldots, C_n = C'$ exists such that, for $i \geq 1$, $C_i$ is obtained from $C_{i-1}$ by replacing a literal $L_i \in C$ with its* c*-factor w.r.t. $C_{i-1}$.*

*A* c-factor *of a set of clauses $N$ is obtained from $N$ by replacing each clause $C \in N$ with an (arbitrary)* c*-factor of $N$. A clause (clause set) is said to be* c-factored *if it is equal to one of its* c*-factors.*

Note that a clause can have several c-factors, depending on the order in which the literals are c-factored. For example, the clause $\neg R(x, y^{\mathsf{c}}) \vee \neg S(x, y^{\mathsf{c}}) \vee T(x, y^{\mathsf{c}})$ has these two c-factors:

$$\neg R(x, z^{\mathsf{c}}) \vee z^{\mathsf{c}} \not\approx_{\mathbf{D}} y^{\mathsf{c}} \vee \neg S(x, y^{\mathsf{c}}) \vee T(x, y^{\mathsf{c}})$$
$$\neg R(x, y^{\mathsf{c}}) \vee \neg S(x, z^{\mathsf{c}}) \vee z^{\mathsf{c}} \not\approx_{\mathbf{D}} y^{\mathsf{c}} \vee T(x, y^{\mathsf{c}})$$

In the remaining sections, we can pick any c-factor of $N$; that is, an arbitrary c-factor of $C$ can be used to replace a clause $C \in N$.

Note that, for a c-factored clause $C$ and for each literal $\neg A \in C$, if $\mathsf{sort}(A|_p) = \mathsf{c}$, then $A|_p$ is a variable that can occur in $C \setminus \{\neg A\}$ only in a positive literal. As shown by the following lemma, c-factoring modifies the set of clauses such that **d**-satisfiability ensures **D**-satisfiability:

**Lemma 6.1.6.** *Let $N$ be a* **D***-extended set of clauses, and $N'$ a* c*-factor of $N$. Then, the following claims hold:*

- *$N$ is $\mathbf{D}$-satisfiable if and only if $N'$ is $\mathbf{D}$-satisfiable.*
- *$N'$ is $\mathbf{d}$-satisfiable if and only if it is $\mathbf{D}$-satisfiable.*

*Proof.* For the first claim, observe that the literal $A$ is equivalent to the following formula:

$$\exists x_1^{\mathsf{c}}, \ldots, x_n^{\mathsf{c}} : A[x_1^{\mathsf{c}}]_{p_1} \ldots [x_n^{\mathsf{c}}]_{p_n} \wedge x_1^{\mathsf{c}} \approx_{\mathbf{D}} A|_{p_1} \wedge \ldots \wedge x_n^{\mathsf{c}} \approx_{\mathbf{D}} A|_{p_n}$$

Moreover, the $\mathsf{c}$-factor of $\neg A$ is obtained from the previous formula using the application of de Morgan laws, so $\neg A$ and its $\mathsf{c}$-factor are equivalent. Therefore, $N$ and $N'$ are equivalent, so they are $\mathbf{D}$-equisatisfiable.

For the second claim, the ($\Leftarrow$) direction follows trivially from the definitions of $\mathbf{D}$- and $\mathbf{d}$-satisfiability. For the ($\Rightarrow$) direction, let $I$ be a classic Herbrand model of $N'$, and $\delta$ an assignment satisfying Definition 6.1.4. If $(I, \delta)$ is not a $\mathbf{D}$-model of $N'$, then there is a clause $C \in N'$ with a ground instance $C\tau$ that is true in $I$, but for which $\delta(C\tau)$ is false in $\delta(I)$. For each positive literal $A \in C$, if $A\tau \in I$, then $\delta(A\tau)$ is true in $\delta(I)$, so $C$ must be of the form $D \vee \neg A_1 \vee \ldots \vee \neg A_n$ where $D\tau$ is false in $I$, and, for all $i$, $A_i\tau \notin I$ but $\delta(A_i\tau)$ is true in $\delta(I)$. The latter is the case only if there are $A_i' \in I$ such that $\delta(A_i') = \delta(A_i\tau)$. Let $p_{ij}$ be positions in $A_i$ such that $\mathsf{sort}(A_i|_{p_{ij}}) = \mathsf{c}$. Since each $A_i$ is $\mathsf{c}$-factored w.r.t. $C$, then, for all $j$, $A_i|_{p_{ij}}$ is a variable $x_{ij}^{\mathsf{c}}$ occurring in $C \setminus \{A_i\}$ only in positive literals. Let $\sigma$ be a ground substitution that is identical to $\tau$ except for the mappings $x_{ij}^{\mathsf{c}} \mapsto A_i'|_{p_{ij}}$. Obviously, $A_i' = A_i\sigma$. Since $I$ is a model of $N'$, $D\sigma \vee \neg A_1\sigma \vee \ldots \vee \neg A_n\sigma$ is true in $I$; this is possible only if $D\sigma$ is true in $I$. Since $D\tau$ is false in $I$, a literal $L' \in D$ must exist such that $L'\sigma$ is true in $I$, but $L'\tau$ is false in $I$. Since, for each $i$, all variables of sort $\mathsf{c}$ from $\neg A_i$ occur only in positive literals of $C$, the truth value of all negative literals in $D\sigma$ and $D\tau$ is identical, so $L'$ must be a positive literal. However, since $\delta(L'\sigma)$ is true in $\delta(I)$ and $\delta(L'\sigma) = \delta(L'\tau)$, we have that $\delta(L'\tau)$ is true in $\delta(I)$, which contradicts the assumption that $\delta(C\tau)$ is false in $\delta(I)$. □

Intuitively, $\mathsf{c}$-factoring allows us to move the concrete terms from literals without concrete predicates into concrete inequalities. In Subsection 6.1.3, we present a procedure for checking $\mathbf{d}$-satisfiability of a set of ground clauses, which we lift to non-ground clauses in Subsections 6.1.4 and 6.1.5. Combined with $\mathsf{c}$-factoring, this yields a refutation procedure for $\mathbf{D}$-satisfiability.

### 6.1.3 Concrete Domain Resolution with Ground Clauses

We now develop the *ground concrete domain resolution* calculus, $\mathcal{G}^{\mathbf{D}}$ for short, for checking $\mathbf{d}$-satisfiability of a set of clauses, where $\mathbf{D}$ is an admissible concrete domain. In order to prevent the presentation from being too technical, we add the *concrete domain resolution* rule to the ordered resolution calculus [13] (see Section 2.4) only, and argue later that the rule can be combined with other calculi as well. As for ordinary resolution, $\mathcal{G}^{\mathbf{D}}$ is parameterized with an admissible ordering $\succ$ on literals, which is extended to clauses by a multiset extension (see Section 2.4).

**Definition 6.1.7.** *Let $S = \{d_i(\mathbf{t_i})\}$ be a set of literals, where $\mathbf{t_i}$ is a vector of terms $t_{i1}, \ldots, t_{ik}$. Then, $\widehat{S}$ is a conjunction $C = \bigwedge d_i(\mathbf{x_i})$, obtained from $S$ by replacing each occurrence of a term with the same variable such that different terms are replaced with distinct variables. For two conjunctions $C_1$ and $C_2$, we write $C_1 \equiv C_2$ if they are equivalent up to variable renaming.*

*A set $S = \{d_i(\mathbf{t_i})\}$ of positive concrete literals is a* **D**-constraint *if $\widehat{S}$ is* not **D**-*satisfiable. A* **D**-*constraint $S$ is* minimal *if $\widehat{S'}$ is* **D**-*satisfiable for each $S' \subsetneq S$; $S$ is* connected *if it cannot be decomposed into two disjoint nonempty subsets $S_1$ and $S_2$ not sharing a common term ($S_1$ and $S_2$ do not share a common term if, for all $d_i(\mathbf{t_i}) \in S_1$ and $d_j(\mathbf{t_j}) \in S_2$, we have $\mathbf{t_i} \cap \mathbf{t_j} = \emptyset$).*

**Lemma 6.1.8.** *Each minimal* **D**-*constraint $S$ is connected.*

*Proof.* Assume that $S$ is a **D**-constraint, but it is not connected. Hence, $S$ can be decomposed into subsets $S_1$ and $S_2$ not sharing a common term. Since $S$ is a minimal **D**-constraint, $\widehat{S_1}$ and $\widehat{S_2}$ are **D**-satisfiable. However, since $\widehat{S_1}$ and $\widehat{S_2}$ do not have a common variable, $\widehat{S_1} \wedge \widehat{S_2} = \widehat{S}$ is **D**-satisfiable as well, which is a contradiction. □

The ground concrete domain resolution calculus $\mathcal{G}^{\mathbf{D}}$ consists of the following inference rules:

**Positive factoring:**
$$\frac{C \vee A \vee \ldots \vee A}{C \vee A}$$

where $(i)$ $A$ is strictly maximal with respect to $C$.

**Ordered resolution:**
$$\frac{C \vee A \quad D \vee \neg A}{C \vee D}$$

where $(i)$ $A$ is strictly maximal with respect to $C$, $(ii)$ $\neg A$ is maximal with respect to $D$.

**Concrete domain resolution:**
$$\frac{C_1 \vee d_1(\mathbf{t_1}) \quad \ldots \quad C_n \vee d_n(\mathbf{t_n})}{C_1 \vee \ldots \vee C_n}$$

where $(i)$ $d_i(\mathbf{t_i})$ are strictly maximal with respect to $C_i$, $(ii)$ $S = \{d_i(\mathbf{t_i})\}$ is a minimal **D**-constraint.

In $\mathcal{G}^{\mathbf{D}}$, the clauses $C \vee A \vee \ldots \vee A$ and $D \vee \neg A$ are called the *main premises*, whereas the clauses $C \vee A$ and $C_i \vee d_1(\mathbf{t_1})$ are called the *side premises*. Note that, under this definition, the concrete domain resolution rule does not have a main premise.

It is well known that effective redundancy elimination criteria are necessary for theorem proving to be applicable in practice. A powerful *standard notion of redundancy* was introduced in [13]. We adapt this notion slightly to take into account that the concrete domain resolution rule does not have a main premise.

**Definition 6.1.9.** *Let $N$ be a set of ground clauses. A ground clause $C$ is* redundant *in $N$ if clauses $D_i \in N$ exist such that $C \succ D_i$ and $D_1, \ldots, D_m \models C$. A ground*

*inference $\xi$ with side premises $C_i$ and a conclusion $D$ is* redundant *in $N$ if clauses $D_i \in N$ exist such that $C_1, \ldots, C_n, D_1, \ldots, D_m \models D$; if $\xi$ has a main premise $C$, then $C \succ D_i$ is required in addition.*

We now prove the soundness and completeness of $\mathcal{G}^{\mathbf{D}}$ under the standard notion of redundancy.

**Lemma 6.1.10** (Soundness)**.** *Let $N$ be a set of ground clauses, $I$ a $\mathbf{d}$-model of $N$, and $N' = N \cup \{C\}$, where $C$ is the conclusion of an inference by $\mathcal{G}^{\mathbf{D}}$ with premises from $N$. Then, $I$ is a $\mathbf{d}$-model of $N'$.*

*Proof.* For an inference by positive factoring or ordered resolution, the claim is trivial and is shown in the same way as in [13]. Let $C$ be obtained by the concrete domain resolution rule, with $S$ being as in the rule definition. Since $S$ is a $\mathbf{D}$-constraint, a $\mathbf{d}$-model $(I, \delta)$ of $N$ can exist only if there is a literal $d_i(\mathbf{t_i}) \in S$ such that $d_i(\mathbf{t_i}) \notin I$. Since $C_i \vee d_i(\mathbf{t_i})$ is by assumption true in $I$, some literal from $C_i$ is true in $I$. Since $C_i \subseteq C$, we have that $C$ is true in $I$ as well. □

**Lemma 6.1.11** (Completeness)**.** *Let $N$ be a set of ground clauses such that each inference by $\mathcal{G}^{\mathbf{D}}$ from premises in $N$ is redundant in $N$. If $N$ does not contain the empty clause, then $N$ is $\mathbf{d}$-satisfiable.*

*Proof.* We extend the model building method from [13] to handle the concrete domain resolution rule. For a set of ground clauses $N$, we define an interpretation $I$ by induction on the clause ordering $\succ$ as follows: for a clause $C$, we set $I_C = \bigcup_{C \succ D} \varepsilon_D$, where $\varepsilon_D = \{A\}$ if $(i)$ $D \in N$, $(ii)$ $D$ is of the form $D' \vee A$ such that $A$ is strictly maximal with respect to $D'$, and $(iii)$ $D$ is false in $I_D$; otherwise, $\varepsilon_D = \emptyset$. Let $I = \bigcup_{D \in N} \varepsilon_D$. A clause $D$ such that $\varepsilon_D = \{A\}$ is called *productive*, and it is said to *produce* the atom $A$ in $I$. Before proving the lemma, we show the following three properties.

Invariant (*): if $C$ is false in $I_D \cup \varepsilon_D$ for $C \preceq D$, then $C$ is false in $I$. Since $C$ is false in $I_D \cup \varepsilon_D$, all negative literals from $C$ are false in $I_D \cup \varepsilon_D$, and, since $I_D \cup \varepsilon_D \subseteq I$, all negative literals from $C$ are false in $I$ as well. Furthermore, since $\neg A \succ A$ and there is no literal between $\neg A$ and $A$, any atom produced by a clause $D' \succ D$ is larger than any literal occurring in $C$, so no clause greater than $C$ can produce an atom that would make $C$ true.

Invariant (**): if $C$ is true in $I_C \cup \varepsilon_C$, then $C$ is true in $I$. If $C$ is true in $I_C \cup \varepsilon_C$ because some positive literal is true in $I_C \cup \varepsilon_C$, since $I_C \cup \varepsilon_C \subseteq I$, the clause $C$ is true in $I$ as well. Otherwise, $C$ is true in $I_C \cup \varepsilon_C$ because some negative literal $\neg A$ is true in $I_C \cup \varepsilon_C$. Since $\neg A \succ A$ and there is no literal between $\neg A$ and $A$, any atom produced by a clause $D \succ C$ is larger than any literal occurring in $C$. Hence, no such clause $D$ can produce $A$, so $\neg A$ is true in $I$ as well. We often use this invariant in its contrapositive form: if $C$ is false in $I$, then it is false in $I_C \cup \varepsilon_C$ as well.

Property (***): if an inference $\xi$ with a main premise $C$, side premises $C_i$, and a conclusion $D$ is redundant in $N$, then there are clauses $D_i \in N$ that are not redundant in $N$ such that $D_1, \ldots, D_m, C_1, \ldots, C_n \models D$ and $C \succ D_i$. If $\xi$ is redundant, by

definition of the standard notion of redundancy, there are clauses $D'_j \in N$ such that $D'_1, \ldots, D'_m, C_1, \ldots, C_n \models D$ and $C \succ D'_i$. Now if some $D'_j$ is redundant, then there are clauses $D''_k$ such that $D''_1, \ldots, D''_{m'} \models D'_j$ and $D'_j \succ D''_k$. Since $\succ$ is well-founded, this process can be continued recursively until we obtain the set of smallest nonredundant clauses for which (***) obviously holds. This property holds analogously if $\xi$ does not have a main premise.

We now prove the claim of Lemma 6.1.11 by contradiction: let us assume that all inferences by $\mathcal{G}^{\mathbf{D}}$ from premises in $N$ are redundant in $N$, but no $\delta$ exists such that $(I, \delta)$ is a **d**-model of $N$. There can be two causes for that:

- There is a clause $C \in N$ that is false in $I$; such a clause is called a *counterexample* for $I$. Since $\succ$ is well-founded and total, we can assume without loss of generality that $C$ is the smallest counterexample. $C$ is obviously not productive, since all productive clauses are true in $I$. $C$ can be nonproductive and false in $I$ if it has one of the following two forms:

  - $C = C' \vee A \vee \ldots \vee A$. Then, $C$ is not productive since $A$ is not strictly maximal in $C$. Since $C$ is false in $I$, by (**) it is false in $I_C \cup \varepsilon_C$. Hence, $C'$ is false in $I_C \cup \varepsilon_C$, and, since $C' \prec C$, by (*) $C'$ is false in $I$. Since $N$ is saturated, the inference by positive factoring resulting in $D = C' \vee A$ is redundant in $N$. $D$ is obviously false in $I$. By the fact that $N$ is saturated and by (***), clauses $D_i \in N$ exist, such that $C \succ D_i$ and $D_1, \ldots, D_n \models D$. Since $C$ is the smallest counterexample, all $D_i$ are true in $I$, but then $D$ is true in $I$ as well, which is a contradiction.
  - $C = C' \vee \neg A$. Since $C$ is false in $I$, we have $A \in I$, where $A$ is produced by a smaller clause $D = D' \vee A$. Similarly as in the previous case, $C'$ is false in $I$. Since $D$ is productive, $D'$ is false in $I_D$, and since $A$ is strictly maximal w.r.t. $D'$, $D'$ is false in $I_D \cup \varepsilon_D$ as well. Since $D' \prec D$, by (*) $D'$ is false in $I$. Since $N$ is saturated, the inference by ordered resolution resulting in $E = C' \vee D'$ is redundant in $N$. $E$ is obviously false in $I$. Because $N$ is saturated and by (***), clauses $D_i \in N$ exist, such that $C \succ D_i$ and $D_1, \ldots, D_n, D' \vee A \models E$. Since $C$ is the smallest counterexample, all $D_i$ are true in $I$, and, since $D' \vee A$ is productive, it is also true in $I$. Hence, $E$ is true in $I$, which is a contradiction.

- All clauses from $N$ are true in $I$, but $I$ contains a set of concrete domain literals $S = \{d_i(\mathbf{t_i})\}$ such that $\widehat{S}$ is **D**-unsatisfiable. We can assume without loss of generality that $S$ is minimal. The literals from $S$ must have been produced by clauses $E_i = C_i \vee d_i(\mathbf{t_i}) \in N$, where $C_i$ is false in $I_{E_i} \cup \varepsilon_{E_i}$. For each $i$, we conclude that $C_i$ is false in $I$ as in the case of ordered resolution. Since $N$ is saturated, the inference by concrete domain resolution from $E_i$ resulting in $D = C_1 \vee \ldots \vee C_n$ is redundant in $N$. Obviously, $D$ is false in $I$. Since the

inference is redundant, by property (***), nonredundant clauses $D_i \in N$ exist such that $D_1, \ldots, D_m, C_1 \vee d_1(\mathbf{t_1}), \ldots, C_n \vee d_n(\mathbf{t_n}) \models D$. All $D_i$ and $C_i \vee d_i(\mathbf{t_i})$ are by assumption true in $I$, implying that $D$ is true in $I$, which is a contradiction.

Hence, an assignment $\delta$ must exist such that $(I, \delta)$ is a **d**-model of $N$ (because we do not consider theories with equality, the first condition of Definition 6.1.2 related to $\approx$ is trivially satisfied), so $N$ is **d**-satisfiable. □

A *fair derivation* by $\mathcal{G}^{\mathbf{D}}$ from a set of clauses $N$ with a limit $N_\infty$ is defined as usual (see Section 2.4). By lemmata 6.1.10 and 6.1.11, we get the following result:

**Theorem 6.1.12.** *The set of ground clauses $N$ is* **d***-unsatisfiable if and only if the limit $N_\infty$ of a fair derivation by $\mathcal{G}^{\mathbf{D}}$ contains the empty clause.*

### 6.1.4 Most General Partitioning Unifiers

Lifting the concrete domain resolution rule to general clauses is not trivial, because unification can only partly guide the rule application. Consider, for example, the set of clauses $N = \{d_1(x_1, y_1), d_2(x_2, y_2)\}$. $N$ has ground instances $d_1(a_1, b_1)$ and $d_2(a_2, b_2)$, which do not share a common term. Hence, a conjunction consisting of these literals is not connected, so, by the contrapositive of Lemma 6.1.8, it cannot be minimal, and the conditions of the concrete domain resolution are not satisfied. However, clauses $d_1(a, b_1)$ and $d_2(a, b_2)$ are also ground instances of $N$, but they do share common terms. Hence, a conjunction of these literals is connected, so the conditions of the concrete domain resolution rule could be satisfied. This is so because it is possible to unify $x_1$ and $x_2$ from $d_1(x_1, y_1)$ and $d_2(x_2, y_2)$. Another possibility is to unify $x_2$ and $y_1$. Even for a set consisting of a single clause, such as $M = \{d(x, y)\}$, it is possible to obtain a **D**-constraint $d(x, x)$ by unifying $x$ and $y$. These examples show that, to check all potential **D**-constraints at the ground level, one has to consider all possible substitutions that produce a minimal **D**-constraint at the nonground level. We formalize this idea in the following definition.

**Definition 6.1.13.** *Let $S = \{d_i(\mathbf{t_i})\}$ be a set of positive concrete domain literals. A substitution $\sigma$ is a* partitioning unifier *of $S$ if the set $S\sigma$ is connected. Furthermore, $\sigma$ is a* most general partitioning unifier *if, for any partitioning unifier $\theta$ such that $\widehat{S\sigma} \equiv \widehat{S\theta}$, a substitution $\eta$ exists such that $\theta = \sigma\eta$. With $\mathsf{MGPU}(S)$ we denote the set of all most general partitioning unifiers of $S$ unique up to variable renaming.*

If no terms from literals in $S$ are unifiable, a most general partitioning unifier of $S$ does not exist. Furthermore, the previous example shows that $S$ can have several most general partitioning unifiers. However, for a given conjunction of concrete literals $\widehat{S\sigma}$, all most general partitioning unifiers are identical up to variable renaming, as demonstrated next.

Let $S = \{d_i(\mathbf{t_i})\}$ be a set of positive concrete domain literals, and $C$ a conjunction over literals in $S$, obtained by replacing terms of each $d_i(\mathbf{t_i})$ with arbitrary variables

(it is not necessary to use different variables in $C$ for different terms from $S$, but the same term in $S$ should always be replaced with the same variable in $C$). For such a $C$, let $m$ be the number of distinct variables in $C$; for each such variable $x_i$, $1 \leq i \leq m$, let $T_{x_i}$ be a list of all terms occurring in a literal in $S$ at a position corresponding to an occurrence of $x_i$ in $C$; finally, let $n = \max |T_{x_i}|$. With $S_C$ we denote the set of terms $t_j = f(s^j_{x_1}, \ldots, s^j_{x_m})$, where $s^j_{x_i}$ is the $j$-th term of $T_{x_i}$ if $j \leq |T_{x_i}|$, and a fresh variable if $j > |T_{x_i}|$, for $1 \leq j \leq n$. Most general partitioning unifiers of $S$ and most general unifiers of $S_C$ are closely related, as shown by the following lemma.

**Lemma 6.1.14.** *Let $\theta$ be a partitioning unifier of a set of concrete domain literals $S = \{d_i(\mathbf{t_i})\}$, and let $C = \widehat{S\theta}$. Then, the substitution $\sigma = \mathsf{MGU}(S_C)$ is the most general partitioning unifier of $S$ such that $\widehat{S\sigma} \equiv C$, and it is unique up to variable renaming.*

*Proof.* To obtain the variable $x_i$ in $C$, $\theta$ must be such that $s^1_{x_i}\theta = \ldots = s^n_{x_i}\theta = T_{x_i}\theta$. Furthermore, $x_i \neq x_j$ for $i \neq j$, so $T_{x_i}\theta \neq T_{x_j}\theta$. Because of the first property, $\theta$ is obviously a unifier of $S_C$, so $\sigma = \mathsf{MGU}(S_C)$ exists, and it is unique up to variable remaining [7]. Furthermore, $\sigma$ is the most general unifier of $S_C$, so a substitution $\eta$ exists such that $\theta = \sigma\eta$. Hence, it is impossible that $T_{x_i}\sigma = T_{x_j}\sigma$ and $T_{x_i}\sigma\eta \neq T_{x_j}\sigma\eta$. Thus, $s^1_{x_i}\sigma = \ldots = s^n_{x_i}\sigma = T_{x_i}\sigma$ and $T_{x_i}\sigma \neq T_{x_j}\sigma$ for $i \neq j$, so $\widehat{S\sigma} \equiv C$. □

**Lemma 6.1.15.** *For a set of concrete domain literals $S = \{d_i(\mathbf{t_i})\}$, $\mathsf{MGPU}(S)$ contains exactly all $\mathsf{MGU}(S_C)$, for each connected conjunction $C$ over literals of $S$.*

*Proof.* For $\sigma \in \mathsf{MGPU}(S)$, $C = \widehat{S\sigma}$ is obviously a connected conjunction over literals of $S$ satisfying conditions of Lemma 6.1.14, so $\sigma$ is equivalent to $\mathsf{MGU}(S_C)$ up to variable renaming. Conversely, let $C$ be a connected conjunction over literals of $S$. Now $\sigma = \mathsf{MGU}(S_C)$ is a partitioning unifier of $S$. Let $C' = \widehat{S\sigma}$. Observe that $C \equiv C'$ does not necessarily hold: it is possible that $T_{x_i}\sigma = T_{x_j}\sigma$ for $i \neq j$. However, $C'$ is a connected conjunction over concrete domain literals satisfying conditions of Lemma 6.1.14, so $\mathsf{MGU}(S_{C'})$ exists, and it is a most general partitioning unifier of $S$. □

Hence, Lemma 6.1.15 gives a brute-force algorithm for computing $\mathsf{MGPU}(S)$: one should systematically enumerate all connected conjunctions $C$ over literals in $S$ and compute $\mathsf{MGU}(S_C)$. The main performance drawback of this algorithm is that all such conjunctions $C$ must be enumerated. For $n$ literals in $S$ of maximal arity $m$, the number of possible assignments of variables in $C$ is bounded by $(nm)^{nm}$, which is obviously exponential. However, for certain logics, it is possible to construct a specialized, but much more efficient algorithm. In Section 6.2, we present such an algorithm applicable to $\mathcal{SHIQ}(\mathbf{D})$.

### 6.1.5 Concrete Domain Resolution with General Clauses

As usual in a resolution setting, we assume that no two premises of an inference rule have any variables in common. The *concrete domain resolution* calculus, $\mathcal{R}^{\mathbf{D}}$ for short, consists of the following rules:

**Positive factoring:**
$$\frac{C \vee A \vee B}{C\sigma \vee A\sigma}$$

where $(i)$ $\sigma = \mathsf{MGU}(A, B)$, $(ii)$ $A\sigma$ is strictly maximal with respect to $C\sigma$.

**Ordered resolution:**
$$\frac{C \vee A \quad D \vee \neg B}{C\sigma \vee D\sigma}$$

where $(i)$ $\sigma = \mathsf{MGU}(A, B)$, $(ii)$ $A\sigma$ is strictly maximal with respect to $C\sigma$, $(iii)$ $\neg B\sigma$ is maximal with respect to $D\sigma$.

**Concrete domain resolution:**
$$\frac{C_1 \vee d_1(\mathbf{t_1}) \quad \ldots \quad C_n \vee d_n(\mathbf{t_n})}{C_1\sigma \vee \ldots \vee C_n\sigma}$$

where $(i)$ clauses $C_i \vee d_i(\mathbf{t_i})$ are not necessarily unique, $(ii)$ for the set $S = \{d_i(\mathbf{t_i})\}$, $\sigma \in \mathsf{MGPU}(S)$, $(iii)$ $d_i(\mathbf{t_i})\sigma$ are strictly maximal with respect to $C_i\sigma$, $(iv)$ $S\sigma$ is a minimal $\mathbf{D}$-constraint.

We briefly comment on Constraint $(i)$ of the concrete domain resolution rule. Consider a set of clauses $N = \{d(f^{\mathsf{c}}(x), b^{\mathsf{c}}), f^{\mathsf{c}}(a) \not\approx_{\mathbf{D}} f^{\mathsf{c}}(c)\}$ and $d^{\mathbf{D}} = \{(1, 2)\}$. Then, $S = \{d(f^{\mathsf{c}}(a), b^{\mathsf{c}}), d(f^{\mathsf{c}}(c), b^{\mathsf{c}}), f^{\mathsf{c}}(a) \not\approx_{\mathbf{D}} f^{\mathsf{c}}(c)\}$ is a set of instances of $N$ such that the conjunction $\widehat{S}$ is $\mathbf{D}$-unsatisfiable. Also, for each proper subset $S' \subset S$, the conjunction $\widehat{S'}$ is $\mathbf{D}$-satisfiable. Hence, unsatisfiability is detected only if several copies of $d(f^{\mathsf{c}}(x), b^{\mathsf{c}})$ are considered simultaneously in the concrete domain resolution rule. Without any assumptions on the nature of the concrete predicates, there is no upper bound on the number of copies that should be considered simultaneously. This property of the calculus obviously leads to undecidability in the general case. To obtain a decision procedure for $\mathcal{SHIQ}(\mathbf{D})$, in Section 6.2 we show that the number of copies that must be considered is bounded.

To prove the completeness of $\mathcal{R}^{\mathbf{D}}$, we show now that, for each ground derivation, there is a corresponding nonground derivation.

**Lemma 6.1.16** (Lifting)**.** *Let $N$ be a set of clauses and $N^G$ the set of ground instances of $N$. For each ground inference $\xi^G$ by $\mathcal{G}^{\mathbf{D}}$ applicable to premises $C_i^G \in N^G$, there is an inference $\xi$ by $\mathcal{R}^{\mathbf{D}}$ applicable to premises $C_i \in N$ such that $\xi^G$ is an instance of $\xi$.*

*Proof.* Let $C_i^G$ be ground premises from $N^G$ participating in $\xi^G$, resulting in a ground clause $D^G$. The ground inference $\xi^G$ of $\mathcal{G}^{\mathbf{D}}$ can be simulated by a corresponding nonground inference $\xi$ where, for each ground premise $C_i^G$, we take the corresponding nonground premise $C_i$. Since all $C_i$ are variable disjoint, there is a ground substitution $\tau$ such that $C_i^G = C_i\tau$. Let us denote with $D$ the result of $\xi$ on $C_i$. We now show that $\xi$ is an inference of $\mathcal{R}^{\mathbf{D}}$.

Let $\xi^G$ be an inference by positive factoring on ground literals $A_i^G$ of $C^G$. Substitution $\tau$ is obviously a unifier for the corresponding nonground literals $A_i$ of $C$. Since any unifier is an instance of the most general unifier $\sigma$ of $A_i$, a substitution $\eta$ exists such that $\tau = \sigma\eta$. Furthermore, if $A_i^G$ is strictly maximal with respect to $C^G$, since $\succ$

is a reduction ordering, the corresponding literal $A_i$ is strictly maximal with respect to $C\sigma$, so $D^G = D\eta$. Hence, $\xi$ is an inference of $\mathcal{R}^{\mathbf{D}}$. Similar reasoning applies in the case of ordered resolution.

Let $\xi^G$ be an inference by concrete domain resolution, and $S = \{d_i(\mathbf{t_i})\}$ be the set of corresponding nonground literals. Since $S\tau$ is a minimal $\mathbf{D}$-constraint, by Lemma 6.1.8, $S\tau$ is connected, so $\tau$ is obviously a partitioning unifier of $S$. Then, by Lemma 6.1.14, a most general partitioning unifier $\sigma$ exists such that $\tau = \sigma\eta$ for some $\eta$ and $\widehat{S\sigma} \equiv \widehat{S\tau}$. Obviously, if $S\tau$ is a minimal $\mathbf{D}$-constraint, so is $S\sigma$. Furthermore, if literals from $S\tau$ are strictly maximal with respect to $C_i^G$, since $\succ$ is a reduction ordering, the corresponding literals from $S\sigma$ are strictly maximal with respect to $C_i\sigma$, so $D^G = D\eta$. Hence, $\xi$ is an inference of $\mathcal{R}^{\mathbf{D}}$. □

The notion of redundancy is lifted to the nonground case as usual [13]: a clause $C$ (an inference $\xi$) is *redundant* in a set of clauses $N$ if all ground instances of $C$ ($\xi$) are redundant in $N^G$. This is enough for soundness and completeness of $\mathcal{R}^{\mathbf{D}}$.

**Theorem 6.1.17.** *A set of clauses $N$ is* $\mathbf{d}$*-unsatisfiable if and only if the limit $N_\infty$ of a fair derivation by $\mathcal{R}^{\mathbf{D}}$ contains the empty clause.*

*Proof.* A set $N$ is $\mathbf{d}$-unsatisfiable if and only if the set of its ground instances $N^G$ is $\mathbf{d}$-unsatisfiable. By Theorem 6.1.12, $N^G$ is $\mathbf{d}$-unsatisfiable if and only if there is a ground derivation $N^G = N_0^G, N_1^G, \ldots, N_\infty^G$, where the limit $N_\infty^G$ contains the empty clause. By Lemma 6.1.16 and the definition of the redundancy for nonground clauses, for each ground derivation, a nonground derivation $N = N_0, N_1, \ldots, N_\infty$ exists, in which each $N_i^G$ is a subset of the set of ground instances of $N_i$. Hence, $N_\infty^G$ contains the empty clause if and only if $N_\infty$ contains the empty clause, so the claim follows. □

$\mathbf{D}$-satisfiability of a set of clauses $N$ can now be checked by the following steps:

- Extend $N$ with $\aleph^{\mathbf{D}}$;
- Compute $N'$—the $\mathsf{c}$-factor of $N \cup \aleph^{\mathbf{D}}$;
- Check $\mathbf{d}$-satisfiability of $N'$ by $\mathcal{R}^{\mathbf{D}}$.

By Lemma 6.1.6, $N'$ is $\mathbf{d}$-satisfiable if and only if $N \cup \aleph^{\mathbf{D}}$ is $\mathbf{D}$-satisfiable, which is the case if and only if $N$ is $\mathbf{D}$-satisfiable by Lemma 6.1.3. A practical problem in applying $\mathcal{R}^{\mathbf{D}}$ to $N'$ is that $\aleph^{\mathbf{D}}$ can be infinite. In general, this obviously leads to undecidability. However, we show in Section 6.2 that, the clauses from $\aleph^{\mathbf{D}}$ are not needed to decide $\mathcal{SHIQ}(\mathbf{D})$.

### 6.1.6 Deleting D-Tautologies

In ordinary resolution, clauses that are true in any model can be removed without jeopardizing completeness of the calculus. Removing such clauses is crucial for practical applicability of resolution, because this drastically reduces the search space. To achieve

the same effect for $\mathcal{R}^{\mathbf{D}}$, we now define the notion of **D**-*tautologies*—clauses that are true in any **D**-model due to properties of the concrete domain. We show that **D**-tautologies can be deleted eagerly in a saturation process while preserving completeness.

**Definition 6.1.18.** *A literal $d(\mathbf{t})$ is a* **D**-*tautology if $\widehat{d(\mathbf{t})} = d(\mathbf{x})$ is satisfied for any assignment $\delta$ of variables $\mathbf{x}$ to elements of $\triangle_{\mathbf{D}}$. A clause is a* **D**-*tautology if it contains a* **D**-*tautology literal.*

For example, the clauses $C \vee f(x) \approx_{\mathbf{D}} f(x)$ and $C \vee \top_{\mathbf{D}}(x)$ are **D**-tautologies. Similarly, if $\triangle_{\mathbf{D}}$ is the set of nonnegative integers, then the clause $C \vee \geq_0 (x)$ is also a **D**-tautology. We now show that such clauses are redundant and can be deleted in a saturation process.

**Lemma 6.1.19.** **D**-*tautologies can be eagerly deleted in a saturation by $\mathcal{R}^{\mathbf{D}}$ without loosing completeness.*

*Proof.* Let $S = \{d_i(\mathbf{t_i})\}$ be a minimal **D**-constraint. Obviously, $S$ cannot contain a **D**-tautology literal $d(\mathbf{t})$, since $S \setminus \{d(\mathbf{t})\}$ would be an even smaller **D**-constraint. Hence, **D**-tautologies do not participate in inferences by the concrete domain resolution rule.

Let $N$ be a set of ground clauses saturated under $\mathcal{G}^{\mathbf{D}}$ not containing the empty clause. Furthermore, let $I$ be a model obtained by applying the model building method from Lemma 6.1.11 to all clauses from $N$ that are not **D**-tautologies, and let $\delta$ be an assignment of elements from $\triangle_{\mathbf{D}}$ to elements of $HU_{\mathtt{c}}$ (such an assignment exists by Lemma 6.1.11, because $N$ is saturated and does not contain the empty clause). Let $I'$ be a model obtained by adding $d(\mathbf{t})$ to $I$ for each **D**-tautology literal $d(\mathbf{t})$ of a ground clause $C \in N$. Since all concrete domain literals occur positively in clauses in $N$, adding concrete literals to $I$ cannot make any clause in $N$ false. For each such literal, we have $\delta(\mathbf{t}) \in d^{\mathbf{D}}$, so $(I', \delta)$ is a Herbrand **d**-model of $N$. Effectively, a **D**-tautology $C$ generates in $I'$ only **D**-tautology concrete literals that cannot participate in a concrete domain resolution rule; thus $C$ can be deleted from $N$.

For the nonground calculus $\mathcal{R}^{\mathbf{D}}$, simply observe that, if a nonground literal $d(\mathbf{t})$ is a **D**-tautology, then for all ground substitutions $\tau$, the literal $d(\mathbf{t})\tau$ is a **D**-tautology as well, so the lifting argument from Lemma 6.1.16 holds without change. □

Note that a clause containing a complementary pair of concrete literals is *not* a **D**-tautology and should not be deleted. Consider the following **d**-unsatisfiable set of clauses (recall that $\approx_{\mathbf{D}}$ and $\not\approx_{\mathbf{D}}$ are just a special concrete domain predicates):

$$\boxed{a \approx_{\mathbf{D}} b} \vee a \approx_{\mathbf{D}} c \tag{6.1}$$

$$\boxed{b \approx_{\mathbf{D}} c} \tag{6.2}$$

$$\boxed{a \not\approx_{\mathbf{D}} b} \vee a \not\approx_{\mathbf{D}} c \tag{6.3}$$

Let $a > b > c > \not\approx_{\mathbf{D}} > \approx_{\mathbf{D}}$; the maximal literal in a clause is denoted $\boxed{\text{like this}}$. Now **d**-unsatisfiability can be demonstrated as follows (the notation $D(xx; yy)$ means that a clause is derived by concrete domain resolution from clauses $xx$ and $yy$):

| | | |
|---|---|---|
| (6.4) | $a \approx_{\mathbf{D}} c \vee \boxed{a \not\approx_{\mathbf{D}} c}$ | D(6.1;6.3) |
| (6.5) | $\boxed{a \approx_{\mathbf{D}} c}$ | D(6.1;6.2;6.4) |
| (6.6) | $\boxed{a \not\approx_{\mathbf{D}} c}$ | D(6.2;6.3;6.5) |
| (6.7) | $\square$ | D(6.5;6.6) |

The clause (6.4) is derived because $\{a \approx_{\mathbf{D}} b, a \not\approx_{\mathbf{D}} b\}$ is a **D**-constraint; the clause (6.5) is derived because $\{a \approx_{\mathbf{D}} b, b \approx_{\mathbf{D}} c, a \not\approx_{\mathbf{D}} c\}$ is a **D**-constraint; the clause (6.6) is derived because $\{b \approx_{\mathbf{D}} c, a \not\approx_{\mathbf{D}} b, a \approx_{\mathbf{D}} c\}$ is a **D**-constraint; and the clause (6.7) is derived because $\{a \approx_{\mathbf{D}} c, a \not\approx_{\mathbf{D}} c\}$ is a **D**-constraint.

Note that (6.4) contains a complementary pair of concrete literals, but it should not be deleted, as the empty clause cannot be derived without it. Intuitively, this clause is needed to produce the literal $a \not\approx_{\mathbf{D}} c$ in the model, which is then used in further applications of concrete domain resolution to detect inconsistency.

### 6.1.7 Combining Concrete Domains with Other Resolution Calculi

The proof of Lemma 6.1.6 is model-theoretic and does not depend on a calculus. Hence, c-factoring can be applied as a preprocessing step, regardless of the calculus.

In order to make the presentation less technical, we considered the concrete domain resolution rule only in combination with ordered resolution. However, from the proof of Lemma 6.1.11, one may see that the concrete domain resolution rule is largely independent from the actual calculus, and can be combined with other calculi whose completeness proof is based on the model generation method [13].

To apply the concrete domain resolution rule to other calculi, the premises of the concrete domain resolution rule must be those clauses that have at least one productive ground instance. For ordered resolution with selection, this means that premises should not contain selected literals (since such clauses do not have productive ground instances). The usual arguments for the calculus under consideration show that, if $N_\infty$ does not contain the empty clause, one can generate an interpretation $I$ using the model generation method such that all clauses from $N_\infty$ are true in $I$. Furthermore, the argument from the second part of Lemma 6.1.11 shows independently that, if all inferences by the concrete domain resolution rule are redundant in $N_\infty$, then there is an assignment $\delta$ such that $(I, \delta)$ is a **d**-model of $N$.

We denote with $\mathcal{BS}^{\mathbf{D}}$ the extension of the basic superposition calculus with a concrete domain resolution rule. We stress that $\approx_{\mathbf{D}}$ is a concrete predicate like any other. Hence, due to encoding of atoms as $\mathcal{E}$-terms (see Section 2.5), a literal $s \circ t$ with $\circ \in \{\approx_{\mathbf{D}}, \not\approx_{\mathbf{D}}\}$ is actually a shortcut for a positive literal $(s \circ t) \approx \mathsf{T}$. Therefore, superposition and reflexivity resolution are not applicable to such literals; rather, they participate only in inferences by concrete domain resolution.

**Theorem 6.1.20.** *Let $N_\infty$ be the set of closures obtained by saturating a set of closures $N$ by $\mathcal{BS}^{\mathbf{D}}$. Then $N$ is $\mathbf{d}$-satisfiable if and only if $N_\infty$ does not contain the empty closure.*

*Proof.* The inference rules of $\mathcal{BS}$ are sound, and the soundness of the concrete domain resolution rule follows exactly as in Lemma 6.1.10. For completeness, let us assume that $N_\infty$ does not contain the empty closure. By [14, 96], it is possible to generate a rewrite system $R_N$, which uniquely defines the Herbrand interpretation ${R_N}^*$ such that ${R_N}^* \models \mathsf{irred}_{R_N}(N)$. Furthermore, by exactly the same argument as in Lemma 6.1.11, a function $\delta$ assigning concrete individuals to concrete terms from $\mathsf{irred}_{R_N}(N)$ exists such that, for each $d(\mathbf{t}) \in {R_N}^*$, we have $\delta(\mathbf{t}) \in d^{\mathbf{D}}$.

However, $(I, \delta)$ is not necessarily a well-formed Herbrand $\mathbf{d}$-model of $\mathsf{irred}_{R_N}(N)$, because it need not satisfy the first condition of Definition 6.1.2: it may be that we have $s_i \approx t_i \in {R_N}^*$ for $1 \leq i \leq n$, but, for some general function symbol $f^{\mathsf{c}}$, we have $\delta(f^{\mathsf{c}}(s_1, \ldots, s_n)) \neq \delta(f^{\mathsf{c}}(t_1, \ldots, t_n))$. However, consider an assignment $\delta'$ defined as $\delta'(f^{\mathsf{c}}(s_1, \ldots, s_n)) = \delta(f^{\mathsf{c}}(\mathsf{nf}_{R_N}(s_1), \ldots, \mathsf{nf}_{R_N}(s_n)))$. Obviously, $\delta'$ satisfies the first condition of Definition 6.1.2. Furthermore, by the definition of ${R_N}^*$, if $d(\mathbf{t}) \in {R_N}^*$, then $d(\mathsf{nf}_{R_N}(\mathbf{t})) \in {R_N}^*$. Because $\delta$ fulfills the second condition of Definition 6.1.2, $\delta'$ fulfills it as well. Hence, $({R_N}^*, \delta')$ is a $\mathbf{d}$-model of $\mathsf{irred}_{R_N}(N)$. Finally, by the standard lifting argument for basic superposition, $({R_N}^*, \delta')$ is a $\mathbf{d}$-model of $N$. □

The proof of Theorem 6.1.20 ensures by a model-theoretic argument that the properties of the model related to equality (the first condition of Definition 6.1.2) are satisfied. To develop an intuition behind this result, we present an alternative, proof-theoretic argument. The first condition of Definition 6.1.2 is obviously fulfilled if, for each general function symbol $f^{\mathsf{c}}$ of arity $n$, we add the following closure:

$$x_1 \not\approx y_1 \vee \ldots \vee x_n \not\approx y_n \vee f^{\mathsf{c}}(x_1, \ldots, x_n) \approx_{\mathbf{D}} f^{\mathsf{c}}(y_1, \ldots, y_n) \tag{6.8}$$

If we select all literals $x_i \not\approx y_i$, such a closure can only participate in a reflexivity resolution inference, producing a closure of the following form:

$$f^{\mathsf{c}}(x_1, \ldots, x_n) \approx_{\mathbf{D}} f^{\mathsf{c}}(x_1, \ldots, x_n) \tag{6.9}$$

Such a closure is obviously a $\mathbf{D}$-tautology, so by Lemma 6.1.19 it can be removed from the closure set. Effectively, the first condition of Definition 6.1.2 is always trivially satisfied; however, it is needed in Definition 6.1.2 to ensure that replacing subterms of concrete terms with equal terms has the usual semantics.

## 6.2 Deciding $\mathcal{SHIQ}(\mathbf{D})$

In this section, we combine the concrete domain resolution rule from Section 6.1 with the algorithm from Chapter 5 to obtain a decision procedure for checking satisfiability of $\mathcal{SHIQ}(\mathbf{D})$ knowledge bases.

Table 6.1: Closures after Preprocessing Stemming from Concrete Datatypes

| | |
|---|---|
| 12 | $\neg T(x, y^{\mathsf{c}}) \vee U(x, y^{\mathsf{c}})$ |
| 13 | $\bigvee(\neg)C_i(x) \vee T(x, f^{\mathsf{c}}(x))$ |
| 14 | $\bigvee(\neg)C_i(x) \vee d(f_1^{\mathsf{c}}(x), \ldots, f_m^{\mathsf{c}}(x))$ |
| 15 | $\bigvee(\neg)C_i(x) \vee f_i^{\mathsf{c}}(x) \not\approx_{\mathbf{D}} f_j^{\mathsf{c}}(x)$ |
| 16 | $\bigvee(\neg)C_i(x) \vee \bigvee_{i=1}^{n} \neg T_i(x, y_i^{\mathsf{c}}) \vee d(y_1^{\mathsf{c}}, \ldots, y_n^{\mathsf{c}})$ |
| 17 | $\bigvee(\neg)C_i(x) \vee \bigvee_{i=1}^{n} \neg T(x, y_i^{\mathsf{c}}) \vee \bigvee_{i=1}^{n} \bigvee_{j=i+1}^{n} y_i^{\mathsf{c}} \approx_{\mathbf{D}} y_j^{\mathsf{c}}$ |
| 18 | $T(a, b^{\mathsf{c}})$ |
| 19 | $\neg T(a, y^{\mathsf{c}}) \vee y^{\mathsf{c}} \not\approx_{\mathbf{D}} b^{\mathsf{c}}$ |
| 20 | $a^{\mathsf{c}} \approx_{\mathbf{D}} b^{\mathsf{c}}$ |
| 21 | $a^{\mathsf{c}} \not\approx_{\mathbf{D}} b^{\mathsf{c}}$ |

The operator $\Omega$ from Section 5.2 can be used to eliminate transitivity axioms from a $\mathcal{SHIQ}(\mathbf{D})$ knowledge base $KB$ by encoding it into an equisatisfiable knowledge base $\Omega(KB)$: concrete roles cannot be transitive, so Theorem 5.2.3 applies without changes. Hence, without loss of generality, in the rest of this section we consider $\mathcal{ALCHIQ}(\mathbf{D})$ knowledge bases only.

With $\mathcal{BS}_{DL}^{\mathbf{D},+}$ we denote the $\mathcal{BS}^{+}$ calculus extended with the concrete domain resolution rule, parameterized as specified in Definition 5.3.3. To obtain a decision procedure for $\mathcal{ALCHIQ}(\mathbf{D})$, we show that the results of Lemma 5.3.6 remain valid when $\mathcal{ALCHIQ}(\mathbf{D})$-closures are saturated under $\mathcal{BS}_{DL}^{\mathbf{D},+}$. In particular, we show that the application of the concrete domain resolution rule does not cause generating terms of arbitrary depth. Furthermore, we show that the maximal length of a $\mathbf{D}$-constraint to be considered is polynomial in $|KB|$, assuming a limit on the arity of concrete predicates, so the complexity of reasoning does not increase.

### 6.2.1 Closures with Concrete Predicates

With $\Xi(KB)$ we denote the set of closures obtained from $KB$ by structural transformation (see Definition 5.3.1) and $\mathsf{c}$-factoring. By definition of $\pi$ from Table 3.1 and Table 3.4, it is easy to see that $\Xi(KB)$ can contain only closures with structure given in Table 5.1, with all variables, function symbols, and predicate arguments being of the sort $\mathsf{a}$, and, additionally, closures with structure given in Table 6.1. Note that closures of type 21 are obtained by applying $\mathsf{c}$-factoring to closures of the form $\neg T(a, b^{\mathsf{c}})$.

We now generalize the closures from Table 6.1 to include closure types produced in a saturation of $\Xi(KB)$ by $\mathcal{BS}_{DL}^{\mathbf{D},+}$: $\mathcal{ALCHIQ}(\mathbf{D})$-*closures* are of the form given in Table 5.2 and Table 6.2. By definition of $\Xi$, it is obvious that Lemma 5.3.2 and Lemma 5.3.4 hold for $\mathcal{ALCHIQ}(\mathbf{D})$-closures as well.

### 6.2.2 Closure of $\mathcal{ALCHIQ}(\mathbf{D})$-Closures under Inferences

We now extend Lemma 5.3.6 to handle $\mathcal{ALCHIQ}(\mathbf{D})$-closures.

**Lemma 6.2.1.** *Let $\Xi(KB) = N_0, \ldots, N_i \cup \{C\}$ be a $\mathcal{BS}_{DL}^{\mathbf{D},+}$-derivation, where $C$ is the conclusion derived from premises in $N_i$. Then, $C$ is either an $\mathcal{ALCHIQ}(\mathbf{D})$-closure, or it is redundant in $N_i$.*

*Proof.* The sorts of the abstract and concrete domain predicates are disjoint; furthermore, in closures of type 11 literals with concrete predicates or concrete equalities are always maximal. Hence, an inference between closures of types 1–7 and 9–13 is not possible. Also, a closure of type 8 can participate in an inference with a closure of type 1–7 only on abstract roles and concepts, and in an inference with a closure of type 9–13 only on concrete roles and concrete literals. Hence, the proof of Lemma 5.3.6 remains valid for closures of types 1–8. Furthermore, decomposition can be applied analogously as in Theorem 5.4.8.

Since equality among concrete terms is actually a concrete predicate, it does not participate in superposition inferences; rather, it participates only in concrete domain resolution. Hence, there are no superposition inferences into closures of type 10, so there are no termination problems as in Lemma 5.3.6. Finally, in a closure of type 8 containing a literal $L = \neg T(\langle a \rangle, y^{\mathsf{c}})$, the literal $L$ is selected. Hence, such a closure can only participate in superposition into the first argument of $L$, or in resolution with a closure of type 3 or 8. In both cases, the result is a closure of type 8.

The most important difference to the proof of Lemma 5.3.6 lies in the application of the concrete domain resolution rule, with $n$ side premises of the form $C_i \vee d_i(\langle \mathbf{t_i} \rangle)$. Let $x_i$ be the free variables of the side premises of type 11, and let $S = \{d_i(\mathbf{t_i})\}$. For any most general partitioning unifier $\sigma$ of $S$, $S\sigma$ must be connected, so there are two possibilities:

- If all side premises are of type 11, $\sigma$ is of the form $\{x_2 \mapsto x_1, \ldots, x_n \mapsto x_1\}$. The result is obviously a closure of type 11.

Table 6.2: Types of $\mathcal{ALCHIQ}(\mathbf{D})$-Closures

| | |
|---|---|
| 8 | in addition to literals from Table 5.2, a closure can additionally contain<br>$\mathbf{T}(\langle \mathbf{a} \rangle, \langle \mathbf{b}^{\mathsf{c}} \rangle) \vee \mathbf{d}(\langle \mathbf{t_1^c} \rangle, \ldots, \langle \mathbf{t_n^c} \rangle) \vee \bigvee \langle t_i^{\mathsf{c}} \rangle \approx_{\mathbf{D}} / \not\approx_{\mathbf{D}} \langle {t^{\mathsf{c}}}_j \rangle \vee \bigvee \neg T(\langle a \rangle, y^{\mathsf{c}}) \vee y^{\mathsf{c}} \not\approx_{\mathbf{D}} b^{\mathsf{c}}$<br>where $t_i^{\mathsf{c}}$ and $t_j^{\mathsf{c}}$ are either a constant $b^{\mathsf{c}}$, or a term $f_i^{\mathsf{c}}([a])$ |
| 9 | $\neg T(x, y^{\mathsf{c}}) \vee U(x, y^{\mathsf{c}})$ |
| 10 | $\mathbf{P}^{\mathbf{f^c}}(x) \vee T(x, \langle f^{\mathsf{c}}(x) \rangle)$ |
| 11 | $\mathbf{P}(x) \vee \mathbf{d}(\langle \mathbf{f_1^c}(x) \rangle, \ldots, \langle \mathbf{f_n^c}(x) \rangle) \vee \bigvee \langle f_i^{\mathsf{c}}(x) \rangle \approx_{\mathbf{D}} / \not\approx_{\mathbf{D}} \langle {f^{\mathsf{c}}}_j(x) \rangle$ |
| 12 | $\mathbf{P}(x) \vee \bigvee_{i=1}^{n} \neg T(x, y_i^{\mathsf{c}}) \vee \bigvee_{i=1}^{n} \overset{n}{}_{j=i+1} y_i^{\mathsf{c}} \approx_{\mathbf{D}} y_j^{\mathsf{c}}$ |
| 13 | $\mathbf{P}(x) \vee \bigvee_{i=1}^{n} \neg T_i(x, y_i^{\mathsf{c}}) \vee d(y_1^{\mathsf{c}}, \ldots, y_n^{\mathsf{c}})$ |

- Assume there is a side premise of type 8, and that $S\sigma$ is connected. If $S\sigma$ would contain a variable $x_i$, then the literal $d_i(\mathbf{f^c_i}(x_i))$ would not contain a ground term, so $S\sigma$ would not be connected. Hence, all literals from $S\sigma$ are ground, and $\sigma$ is of the form $\{x_1 \mapsto c_1, \ldots, x_n \mapsto c_n\}$. The result is a closure of type 8.

Hence, all nonredundant inferences of $\mathcal{BS}^{\mathbf{D},+}_{DL}$ applied to $\mathcal{ALCHIQ}(\mathbf{D})$-closures produce an $\mathcal{ALCHIQ}(\mathbf{D})$-closure. □

By inspecting the proof of Lemma 6.2.1, one may easily extend the Corollary 5.3.7 to include $\mathcal{ALCHIQ}(\mathbf{D})$ closures:

**Corollary 6.2.2.** *If a closure of type 8 participates in a $\mathcal{BS}^{\mathbf{D},+}_{DL}$ inference in a derivation from Lemma 6.2.1, the unifier $\sigma$ contains only ground mappings, and the conclusion is a closure of type 8. Furthermore, a closure of type 8 cannot participate in an inference with a closure of type 4 or 6.*

The proof of Lemma 6.2.1 also shows that closures from $\aleph^{\mathbf{D}}$ cannot participate in inferences by concrete domain resolution. Namely, in nonground inferences, an individual $a_\alpha$ does not unify with $f^{\mathbf{c}}(x)$, and for ground inferences, an individual $a_\alpha$ does not occur in any ABox clause. Hence, it is not necessary to ensure that $\Xi(KB)$ is $\mathbf{D}$-extended.

**Corollary 6.2.3.** *In any $\mathcal{BS}^{\mathbf{D},+}_{DL}$ derivation from $\Xi(KB) \cup \aleph^{\mathbf{D}}$, the closures from $\aleph^{\mathbf{D}}$ do not participate in any inferences.*

### 6.2.3 Termination and Complexity Analysis

Establishing termination and determining the complexity is slightly more difficult. Let $m$ denote the maximal arity of a concrete domain predicate, $d$ the number of concrete domain predicates, and $f$ the number of function symbols. By skolemizing $\exists T_1, \ldots, T_m.d$, we introduce $m$ function symbols, so $f$ is linear in $|KB|$ for unary coding of numbers as in Lemma 5.3.9. In addition to literals from $\mathcal{ALCHIQ}$-closures, $\mathcal{ALCHIQ}(\mathbf{D})$-closures can contain literals of the form $d(\langle f_1(x)\rangle, \ldots, \langle f_m(x)\rangle)$. The maximal nonground closure contains all such literals, of which there can be $d(2f)^m$ many (the factor 2 takes into account that each term can be marked or not); moreover, there are $2^{d(2f)^m}$ different combinations of concrete domain literals. This presents us with a problem: in general, $m$ is linear in $|KB|$, so the number of closures is doubly-exponential, thus invalidating Lemma 5.3.9.

A possible solution to this problem is to assume a bound on the arity of concrete predicates. This is justifiable from a practical point of view: it is hard to imagine a practically useful concrete domain with predicates of unbounded arity. Thus, $m$ becomes a constant, and does not depend on $|KB|$. The maximal length of a closure is now polynomial in $|KB|$, and the number of closures is exponential in $|KB|$ in the same way as in Lemma 5.3.9. The following lemma provides the last step necessary to determine the complexity of the algorithm.

**Lemma 6.2.4.** *The maximal number of side premises participating in a concrete domain resolution rule in a $\mathcal{BS}_{DL}^{\mathbf{D},+}$ derivation from Lemma 6.2.1 is at most polynomial in $|KB|$, assuming a bound on the arity of concrete predicates.*

*Proof.* Let $S = \{d_i(\mathbf{t_i})\}$ be the multiset of maximal concrete domain literals of $n$ side premises $C_i \vee d_i(\mathbf{t_i})$, and let $\sigma$ be a most general partitioning unifier of $S$. Obviously, $S\sigma$ should not contain repeated literals $d_i(\mathbf{t_i})\sigma$; otherwise, $\widehat{S\sigma}$ contains repeated conjuncts and is not minimal. Hence, the longest set $S\sigma$ is the one in which each $d_i(\mathbf{t_i})\sigma$ is distinct.

Let $m$ be the maximal arity of a concrete domain predicate, $f$ the number of function symbols, $d$ the number of concrete domain predicates, and $c$ the number of constants in the signature of $\Xi(KB)$. Then, there are at most $\ell_{ng} = df^m$ distinct nonground concrete domain literals, and at most $\ell_g = d(c + cf)^m$ distinct ground concrete domain literals. Assuming a bound on $m$ and for unary coding of numbers, both $\ell_{ng}$ and $\ell_g$ are polynomial in $|KB|$.

If all side premises are of type 11, the maximal literals are not ground. Since $S\sigma$ should be connected, the only possible form that $\sigma$ can take is $\{x_2 \mapsto x_1, \ldots, x_n \mapsto x_1\}$. Hence, $S\sigma$ contains only one variable $x_1$, so the maximal number of distinct literals in $S\sigma$ is $\ell_{ng}$. Thus, the maximal number of nonground side premises $n$ to be considered is bounded by $\ell_{ng}$, and all side premises are unique up to variable renaming.

If there is a side premise of type 8, at least one maximal literal is ground. Since $S\sigma$ should be connected, $\sigma$ can be of the form $\{x_1 \mapsto c_1, \ldots, x_n \mapsto c_n\}$. All literals in $S\sigma$ are ground, so the maximum number of distinct literals in $S\sigma$ is $\ell_g$. Thus, the maximal number of side premises $n$ (by counting each possible copy of a premise separately) to be considered is bounded by $\ell_g$. □

**Theorem 6.2.5.** *Let $KB$ be an $\mathcal{ALCHIQ}(\mathbf{D})$ knowledge base, defined over an admissible concrete domain $\mathbf{D}$, for which $\mathbf{D}$-satisfiability of finite conjunctions over $\Phi_{\mathbf{D}}$ can be decided in deterministic exponential time. Then, saturation of $\Xi(KB)$ by $\mathcal{BS}_{DL}^{\mathbf{D},+}$ with eager application of redundancy elimination rules decides satisfiability of $KB$, and runs in time exponential in $|KB|$, for unary coding of numbers and assuming a bound on the arity of concrete predicates.*

*Proof.* As already explained, a polynomial bound on the length, and an exponential bound on the number of $\mathcal{ALCHIQ}(\mathbf{D})$-closures follows in the same way as in Lemma 5.3.9 and Theorem 5.4.8. By substituting Lemma 6.2.1 for Lemma 5.3.6, the proof of the Theorem 5.3.10 also holds in the case of $\mathcal{ALCHIQ}(\mathbf{D})$ knowledge bases for all inferences apart from the concrete domain resolution rule. The only remaining problem is to show that, in applying the concrete domain resolution rule, the number of satisfiability checks of conjunctions over $\mathbf{D}$ is exponential in $|KB|$, and that the length of each conjunction is polynomial in $|KB|$.

To apply the concrete domain resolution rule to a set of closures $N$, a subset $N' \subseteq N$ must be selected, for which maximal concrete domain literals are unique up to variable renaming. By Lemma 6.2.4, $\ell = |N'|$ is polynomial in $|KB|$, and $N'$

contains at most $\ell$ variables. If $N'$ does not contain a closure of type 8, there is exactly one substitution $\sigma$ that unifies all $\ell$ variables. If there is at least one closure of type 8, then each one of $\ell$ variables can be assigned to one of $c$ constants, producing $c^\ell$ combinations, which is exponential in $|KB|$. Closures in $N'$ are chosen from the maximal set of all closures, which is exponential in $|KB|$, and since $|N'|$ is polynomial in $|KB|$, the number of different sets $N'$ is exponential in $|KB|$. Hence, the maximal number of **D**-constraints that should be examined by the concrete domain resolution rule is exponential in $|KB|$, where the length of each **D**-constraint is polynomial in $|KB|$. Under the assumption that the satisfiability of each **D**-constraint can be checked in deterministic exponential time, all inferences by the concrete domain resolution rule can be performed in exponential time, so the claim of the theorem follows. □

The proof of Lemma 6.2.4 demonstrates how to implement the concrete domain resolution rule in practice. There are two cases:

- For concrete domain resolution without a side premise of type 8, the most general partitioning unifier is of the form $\sigma = \{x_2 \mapsto x_1, \ldots, x_n \mapsto x_1\}$. Hence, we should consider only closures with maximal literals unique up to variable renaming, without several copies of one and the same closure.
- For concrete domain resolution where at least one side premise is of type 8, we first choose a connected set of closures ${\Delta^g}_0$ of type 8. Let ${\Delta^c}_0$ be the set of constants, and ${\Delta^f}_0$ the set of function symbols occurring in a ground term $f(c)$ in a closure from ${\Delta^g}_0$. We then select the set of closures ${\Delta^{ng}}_0$ of type 11 containing a function symbol from ${\Delta^f}_0$. To obtain a connected constraint, a most general partitioning unifier $\sigma$ of ${\Delta^g}_0 \cup {\Delta^{ng}}_0$ contains mappings of the form $x_i \mapsto c$, for $c \in \Delta^c_0$. To make all literals in the constraint unique, a closure from ${\Delta^{ng}}_0$ can be used at most $|\Delta^c_0|$ times. For each such unifier $\sigma$, let ${\Delta^g}_1 = {\Delta^g}_0\sigma \cup {\Delta^{ng}}_0$. It is possible that, for ${\Delta^f}_1$ the set of function symbols occurring in a ground term $f(c)$ in ${\Delta^g}_1$, we have ${\Delta^f}_0 \subsetneq {\Delta^f}_1$; in this case we repeat the process iteratively. The process will terminate, since the set of closures is finite, and will yield a maximal possible set of connected concrete literals. A minimal connected constraint is then a subset of this maximal literal set.

## 6.3 Example

To better understand how to apply the concrete domain resolution rule in practice, we apply $\mathcal{BS}^{\mathbf{D},+}_{DL}$ to the example from Subsection 3.2.3, and show proof-theoretically that $KB_3 \cup KB_4 \models_{\mathbf{D}} GrWS(pc_2)$. This is the case if and only if $KB'$ is unsatisfiable, where $KB'$ is defined as follows:

$$KB' = KB_3 \cup KB_4 \cup \{\neg GrWS(pc_2)\} \tag{6.10}$$

**Translation into Closures.** Note that, in Subsection 3.2.3, the knowledge base $KB_3$ is defined as $KB_1$ form Subsection 3.1.1, extended with axioms (3.18)–(3.19). The translation of $KB_1$ into closures has been presented in Section 5.5, and it consists of closures (5.34)–(5.53); we now translate the axioms (3.18)–(3.19) from $KB_4$.

The axiom (3.19) contains a nonatomic subconcept $PC \sqcap \exists price.{\geq_{2000}}$, so we introduce a new concept $Q_4$, and replace (3.19) with (6.11) and (6.12). Furthermore, (3.21) contains a nonliteral concept $\exists price.{\leq_{1500}}$, so we introduce a new concept $Q_5$, and replace (3.21) with (6.13) and (6.14).

$$HighEnd \sqsubseteq GrWS \sqcup Q_4 \tag{6.11}$$

$$Q_4 \sqsubseteq PC \sqcap \exists price.{\geq_{2000}} \tag{6.12}$$

$$Q_5 \sqsubseteq \exists price.{\leq_{1500}} \tag{6.13}$$

$$Q_5(pc_2) \tag{6.14}$$

The clausification algorithm can now be applied directly, yielding the following closures:

$$\neg price(x, y_1) \vee \neg price(x, y_2) \vee y_1 \approx_{\mathbf{D}} y_2 \tag{6.15}\ (3.18)$$

$$\neg HighEnd(x) \vee GrWS(x) \vee Q_4(x) \tag{6.16}\ (6.11)$$

$$\neg Q_4(x) \vee PC(x) \tag{6.17}\ (6.12)$$

$$\neg Q_4(x) \vee price(x, h(x)) \tag{6.18}\ (6.12)$$

$$\neg Q_4(x) \vee {\geq_{2000}}(h(x)) \tag{6.19}\ (6.12)$$

$$\neg Q_5(x) \vee price(x, i(x)) \tag{6.20}\ (6.13)$$

$$\neg Q_5(x) \vee {\leq_{1500}}(i(x)) \tag{6.21}\ (6.13)$$

$$Q_5(pc_2) \tag{6.22}\ (6.14)$$

$$HighEnd(pc_2) \tag{6.23}\ (3.20)$$

$$\neg GrWS(pc_2) \tag{6.24}\ (6.10)$$

**Saturation by $\mathcal{BS}^{\mathbf{D},+}_{DL}$.** The $\mathcal{E}$-term ordering, the selection function, and the notation are the same as in Section 5.5. D(...) denotes the application of the concrete domain resolution rule. We now show the inference steps in saturating $\Xi(\Omega(KB'))$ by $\mathcal{BS}^{\mathbf{D},+}_{DL}$.

$\Rightarrow$ (6.25) $\neg Q_4(x) \vee \boxed{{\geq_{2000}}(h(x))}$

$\Rightarrow$ (6.26) $\neg Q_5(x) \vee \boxed{{\leq_{1500}}(i(x))}$

$\Rightarrow$ (6.27) $\neg Q_4(x) \vee \boxed{price(x, h(x))}$

$\Rightarrow$ (6.28) $\neg Q_5(x) \vee \boxed{price(x, i(x))}$

$\Rightarrow$ (6.29) $\boxed{\neg price(x, y_1)} \vee \boxed{\neg price(x, y_2)} \vee y_1 \approx_{\mathbf{D}} y_2$

R(6.27;6.28) $\neg Q_4(x) \vee \neg Q_5(x) \vee [h(x)] \approx_{\mathbf{D}} [i(x)]$

$\Rightarrow$ (6.30) $\neg Q_4(x) \vee \neg Q_5(x) \vee \boxed{[h(x)] \approx_{\mathbf{D}} [i(x)]}$

D(6.25;6.26) $\neg Q_4(x) \vee \neg Q_5(x)$

We briefly explain the last inference. The maximal literals of closures (6.25), (6.26) and (6.30) are $S = \{\geq_{2000}(h(x)), \leq_{1500}(i(x_1)), h(x_2) \approx_{\mathbf{D}} i(x_2)\}$ (since the premises of each inference rule are assumed to be variable disjoint, the variables in closures (6.25) and (6.26) are renamed appropriately). The substitution $\sigma = \{x_1 \mapsto x, x_2 \mapsto x\}$ is the only MGPU of $S$, and $S\sigma = \{\geq_{2000}(h(x)), \leq_{1500}(i(x)), h(x) \approx_{\mathbf{D}} i(x)\}$. The conjunction $\widehat{S\sigma} = \geq_{2000}(y) \wedge \leq_{1500}(z) \vee y \approx_{\mathbf{D}} z$ is obtained from $S\sigma$ by replacing the term $h(x)$ with $y$, and the term $i(x)$ with $z$. Obviously, $\widehat{S\sigma}$ is **D**-unsatisfiable, so we apply the concrete domain resolution rule. We now continue the saturation.

$\Rightarrow$ (6.31) $\boxed{HighEnd(pc_2)}$

$\Rightarrow$ (6.32) $\boxed{\neg HighEnd(x)} \vee GrWS(x) \vee Q_4(x)$

R(6.31) $GrWS([pc_2]) \vee Q_4([pc_2])$

$\Rightarrow$ (6.33) $\boxed{GrWS([pc_2])} \vee Q_4([pc_2])$

$\Rightarrow$ (6.34) $\boxed{\neg GrWS(pc_2)}$

R(6.33) $Q_4([pc_2])$

$\Rightarrow$ (6.35) $\neg Q_4(x) \vee \boxed{\neg Q_5(x)}$

$\Rightarrow$ (6.36) $\boxed{Q_5(pc_2)}$

R(6.35) $\neg Q_4([pc_2])$

$\Rightarrow$ (6.37) $\boxed{Q_4([pc_2])}$

$\Rightarrow$ (6.38) $\boxed{\neg Q_4([pc_2])}$

R(6.37) $\square$

$\Rightarrow$ (6.39) $\square$

In (6.39), the empty closure is added to $W$, so $\Xi(\Omega(KB'))$ is **d**-unsatisfiable. Also, $\Xi(\Omega(KB'))$ is $\mathsf{c}$-factored (it does not contain a negative literal containing a concrete term), so it is **D**-unsatisfiable as well, thus implying $KB_3 \cup KB_4 \models_{\mathbf{D}} GrWS(pc_2)$.

## 6.4 Related Work

The need to represent and reason with concrete data in description logic systems was recognized early on. Early systems, such as MESON [40] and CLASSIC [22], provided such features, mostly in an ad-hoc manner by means of built-in predicates.

The first rigorous treatment of reasoning with concrete data in description logics was presented by Baader and Hanschke in [5]. The authors introduce the notion of a concrete domain and consider reasoning in $\mathcal{ALC}(\mathcal{D})$, a logic allowing feature chains (chains of functional roles) to occur in existential and universal restrictions. The authors show that adding a concrete domain does not increase the complexity of checking concept satisfiability—that is, it remains in PSPACE. Sound and complete reasoning can be performed by extending the standard tableau algorithm for $\mathcal{ALC}$ with an additional branch closure check, which detects potential unsatisfiability of concrete domain constraints on a branch. Furthermore, if transitive closure of roles is allowed, the authors show that checking concept satisfiability becomes undecidable. The approach of Baader and Hanschke was implemented in the TAXON system [63].

Contrary to modern description logic systems, the approach presented in [5] does not allow TBoxes. In [89] it was shown that allowing cyclic TBoxes makes reasoning with a concrete domain undecidable in general. More important, undecidability holds for all numeric concrete domains, which are probably the most practically relevant ones. Still, in [87] it was shown that reasoning with a temporal concrete domain is decidable even with cyclic TBoxes, thus providing for an expressive description logic with features for temporal modeling.

It turned out that even without cyclic TBoxes, extending the logic is difficult. In [89] it was shown that extending $\mathcal{ALC}(\mathcal{D})$ with either acyclic TBoxes, inverse roles, or role-forming concrete domain constructor makes reasoning NEXPTIME-hard. The main reason for this is the presence of the feature chain concept constructor, which can be used to force equality of objects at the end of two distinct chains of features in a model. Hence, feature chains with concrete domains destroy the tree-model property, which was identified in [147] as the main explanation for the good properties of modal and description logics.

Since obtaining expressive logics with concrete domains turned out to be difficult, another path to extending the logic was taken in [62], by prohibiting feature chains. In this way, concrete domain reasoning is restricted only to immediate successors of an abstract individual, so the tree-model property remains preserved. A decision procedure for an expressive $\mathcal{ALCNH}_{R^+}$ logic extended with concrete domains without feature chains was presented in [62], obtained by extending the tableau algorithm with concrete domain constraint checking. In [70], the term *datatypes* was introduced for a concrete domain without feature chains, and a tableau decision procedure for $\mathcal{SHOQ}(\mathbf{D})$ (a logic providing nominals and datatypes) was presented. This approach was extended to allow for $n$-ary concrete roles in [101]. The results of this research influenced significantly the development of the Semantic Web ontology language OWL [106], which also supports datatypes.

Apart from fundamental issues, such as decidability and complexity or reasoning, other issues were considered to make concrete domains practically applicable. In practice, usually several different datatypes are needed. For example, an application might use the string datatype to represent a person's name, and the integer datatype to represent a person's age. It is natural to model each datatype as a separate concrete

domain, and then to integrate several concrete domains for reasoning purposes. An approach for integrating concrete domains was already presented in [5], and was further extended to *datatype groups* in [103], which simplify the integration process by taking care of the extensions of negative concrete literals.

Until now, all existing approaches consider reasoning with a concrete domain in tableau and automata frameworks. In the resolution setting, many Prolog-like systems, such as XSB,[1] provide built-in predicates to handle datatypes. However, to the best of our knowledge, none of these approaches is complete for even the basic logic $\mathcal{ALC}(\mathcal{D})$. Built-in predicates are only capable of handling explicit datatype constants, and cannot cope with individuals introduced by existential quantification. Hence, ours is the first approach that we are aware of that supports reasoning with a concrete domain in the resolution setting and is complete w.r.t. the semantics from [5].

Concrete domain resolution calculus can be viewed as an instance of theory resolution [139]. In fact, it is similar to ordered theory resolution [15], in which inferences with theory literals are restricted to maximal literals only. It has been argued in [15] that tautology deletion is not compatible with ordered theory resolution. However, the concrete domain resolution calculus is fully compatible with the standard notion of redundancy, so all existing deletion and simplification techniques can be freely used. Furthermore, in our work we explicitly address the issues specific to concrete domains, such as the necessity to consider several copies of a clause in an inference by the concrete domain resolution ruls.

[1] `http://xsb.sourceforge.net/`

# Chapter 7

# Reducing Description Logics to Disjunctive Datalog

Based on the algorithms presented in Chapters 5 and 6, we now develop an algorithm for reducing a $\mathcal{SHIQ}(\mathbf{D})$ knowledge base $KB$ to a disjunctive datalog program $\mathsf{DD}(KB)$. The program $\mathsf{DD}(KB)$ entails the same set of ground facts as $KB$, so it can be used to answer queries in $KB$. Furthermore, we also present an algorithm for answering queries in $\mathsf{DD}(KB)$ that runs in exponential time, and thus show that our approach does not increase the complexity of reasoning.

For the algorithms in this chapter, the distinction between the skeleton and the substitution part of a closure is not important. Hence, instead of "closure," we use a more common term "clause."

## 7.1 Overview

Our reduction algorithm is based on the observation that, while saturating $\Xi(KB)$ using $\mathcal{BS}_{DL}^{\mathbf{D},+}$, after performing all inferences with nonground clauses, all remaining inferences involve a clause of type 8 containing terms whose depth is at most one. Intuitively, we simulate these terms with fresh constants, and thus allow transforming each $\mathcal{BS}_{DL}^{\mathbf{D},+}$ refutation from $\Xi(KB)$ to a refutation in $\mathsf{DD}(KB)$. The reduction algorithm consists of the following steps:

- Transitivity axioms are eliminated using the transformation from Section 5.2. Hence, we consider only $\mathcal{ALCHIQ}(\mathbf{D})$ knowledge bases in the remaining sections.
- The TBox and the RBox clauses of $\Xi(KB)$ are saturated together with the clauses from $\mathsf{gen}(KB)$ (see Definition 5.4.9) using $\mathcal{BS}_{DL}^{\mathbf{D},+}$, thus performing all inferences with nonground clauses. As shown in Lemma 7.2.1, certain clauses can be removed from the saturated set, as they cannot participate in further inferences.
- If the saturated set does not contain the empty clause, function symbols are eliminated from the clauses as explained in Section 7.2. Lemma 7.2.4 demon-

strates that this transformation does not affect satisfiability. Intuitively, if ABox clauses are added to the saturated set and saturation is continued, all remaining inferences involve a clause of type 8 and produce a clause of type 8, and each such inference can be simulated in the function-free clause set.

- In order to reduce the size of the datalog program, some irrelevant clauses are removed as explained in Section 7.3. Lemma 7.3.2 demonstrates that this transformation also does not affect satisfiability.
- Transformation of $KB$ into $\mathsf{DD}(KB)$ is explained in Section 7.4 and is straightforward: it suffices to transform each clause into the equivalent sequent form. Theorem 7.4.2 summarizes the properties of the resulting datalog program. As discussed in Subsection 4.8.2, such a program can be interpreted under minimal or standard first-order semantics, as long as it is used only for answering positive ground queries.

## 7.2 Eliminating Function Symbols

Let $KB$ be an extensionally reduced $\mathcal{ALCHIQ}(\mathbf{D})$ knowledge base $KB$; furthermore, let $\Gamma_{\mathcal{TR}g} = \Xi(KB_{\mathcal{T}} \cup KB_{\mathcal{R}}) \cup \mathsf{gen}(KB)$. By the definitions of $\Xi$ and $\mathsf{gen}$, the set $\Gamma_{\mathcal{TR}g}$ does not contain clauses of type 8. Furthermore, let $\Gamma = \mathsf{Sat}_{\mathsf{R}}(\Gamma_{\mathcal{TR}g}) \cup \Xi(KB_{\mathcal{A}})$, where $\mathsf{Sat}_{\mathsf{R}}(\Gamma_{\mathcal{TR}g})$ is the *relevant set of saturated clauses*—that is, clauses of types 1, 2, 3, 5, 7, and 9–13 obtained by saturating $\Gamma_{\mathcal{TR}g}$ by $\mathcal{BS}_{DL}^{\mathbf{D},+}$ with eager application of redundancy elimination rules. Intuitively, $\mathsf{Sat}(\Gamma_{\mathcal{TR}g})$ contains all nonredundant clauses derivable by $\mathcal{BS}_{DL}^{\mathbf{D},+}$ from the TBox and the RBox, so each further inference in the saturation of $\Gamma$ involves a clause of type 8. Such a clause cannot participate in an inference with a clause of type 4 or 6, so we can safely delete the latter clauses, and consider only the $\mathsf{Sat}_{\mathsf{R}}(\Gamma_{\mathcal{TR}g})$ subset. Adding $\mathsf{gen}(KB)$ is necessary to compute all nonground consequences of clauses possibly introduced by decomposition.

**Lemma 7.2.1.** *$KB$ is $\mathbf{D}$-unsatisfiable if and only if $\Gamma$ is $\mathbf{D}$-unsatisfiable, for an extensionally reduced $\mathcal{ALCHIQ}(\mathbf{D})$ knowledge base $KB$,*

*Proof.* Let $\Gamma' = \Xi(KB) \cup \mathsf{gen}(KB)$. Because it does not contain clauses of type 8, $\Gamma'$ is $\mathsf{c}$-factored. Since all clauses from $\mathsf{gen}(KB)$ contain a new predicate $Q_{R,f}$, any interpretation of $\Xi(KB)$ can be extended to an interpretation of $\Gamma'$ by adjusting the interpretation of $Q_{R,f}$ as needed. Hence, $KB$ is $\mathbf{D}$-equisatisfiable with $\Xi(KB)$ by Lemma 5.3.2; $\Xi(KB)$ is $\mathbf{D}$-equisatisfiable with $\Gamma'$; and, by Lemma 6.1.6 and Theorem 6.1.20, $\Gamma'$ is $\mathbf{D}$-unsatisfiable if and only if the set of clauses derived by saturating $\Gamma'$ with $\mathcal{BS}_{DL}^{\mathbf{D},+}$ contains the empty clause. Since the order in which the inferences are performed in a derivation $\Gamma' = N_0, N_1, \ldots$ can be chosen don't-care nondeterministically, we perform all nonredundant inferences among clauses from $\Gamma_{\mathcal{TR}g}$ first. Let $N_i = \mathsf{Sat}(\Gamma_{\mathcal{TR}g}) \cup \Xi(KB_{\mathcal{A}})$ be the resulting intermediate set of clauses.

If $N_i$ contains the empty clause, $\Gamma$ contains it as well (the empty clause is of type 5), and the claim of the lemma follows. Otherwise, we continue the saturation of $N_i$. In such a derivation, each $N_j, j > i$, is obtained from $N_{j-1}$ by an inference involving at least one clause not in $N_i$. By induction on the derivation length, one can show that $N_j \setminus N_i$ contains only clauses of type 8: namely, all nonredundant inferences between clauses of type other than 8 have been performed in $N_i$, and, by Corollary 6.2.2, each inference involving a clause of type 8 produces a clause of type 8. A conclusion of an inference can be decomposed into a clause of type 8 and a clause of type 3; however, the resulting clause of type 3 has the form $\neg Q_{R,f}(x) \vee R(x, [f(x)])$ and is, by Lemma 5.4.10, contained in $\mathsf{gen}(KB)$, so only a clause of type 8 is added to $N_j$.

Furthermore, by Corollary 6.2.2, a clause of type 4 and 6 can never participate in an inference with a clause of type 8. Hence, such a clause cannot be used to derive a clause in $N_j \setminus N_i$, $j > i$, so $N_i$ can safely be replaced by $\Gamma$. Any set of clauses $N_j$, $j > i$, that can be obtained by saturation from $\Gamma'$, can be obtained by saturation from $\Gamma$ as well, modulo clauses of type 4 and 6. Hence, the saturation of $\Gamma'$ by $\mathcal{BS}_{DL}^{\mathbf{D},+}$ derives the empty clause if and only if the saturation of $\Gamma$ by $\mathcal{BS}_{DL}^{\mathbf{D},+}$ derives the empty clause, so the claim of the lemma follows. □

If $KB$ does not use number restrictions, further optimizations are possible. By Corollary 5.3.8, clauses of types 3 or 5 containing a function symbol cannot participate in an inference with a clause of type 8. Hence, such clauses can also be eliminated from the saturated set; that is, they need not be included in $\mathsf{Sat}_{\mathsf{R}}(\Gamma_{\mathcal{TR}g})$. Observe that this does not necessarily hold for clauses of type 10 and 11; namely, if there is a clause of type 13 with more than one literal $\neg T_i(x_i, y_i^{\mathsf{c}})$, then such a clause might derive a ground concrete literal containing a function symbol.

We now show how to eliminate function symbols from clauses in $\Gamma$. Intuitively, the idea is to replace each ground term $f(a)$ with a new constant, denoted with $a_f$. For each function symbol $f$ we introduce a new predicate symbol $S_f$ that contains, for each abstract constant $a$, a tuple of the form $S_f(a, a_f)$. Thus, $S_f$ contains the $f$-successor of each constant. A clause $C$ is transformed by replacing each occurrence of a term $f(x)$ with a new variable $x_f$, and appending the literal $\neg S_f(x, x_f)$. The Herbrand universe of the transformed clause set is finite and can be enumerated in a finite predicate $HU$; this predicate is used to bind unsafe variables in clauses. This transformation is formalized in the following definition:

**Definition 7.2.2.** *The operator $\lambda$ maps terms to terms as follows:*

- $\lambda(a) = a$;
- $\lambda(f(a)) = a_f$, *for $a_f$ a new globally unique constant*[1] *with* $\mathsf{sort}(a_f) = \mathsf{sort}(f(a))$;
- $\lambda(x) = x$;
- $\lambda(f(x)) = x_f$, *for $x_f$ a new globally unique variable with* $\mathsf{sort}(x_f) = \mathsf{sort}(f(x))$.

[1] Globally unique means that, for some $f$ and $a$, the constant $a_f$ is uniquely defined.

*We extend $\lambda$ to $\mathcal{ALCHIQ}(\mathbf{D})$-clauses such that, for an $\mathcal{ALCHIQ}(\mathbf{D})$-clause $C$, $\lambda(C)$ gives a function-free clause as follows:*

1. *Each term $t$ in the clause is replaced with $\lambda(t)$.*
2. *For each variable $x_f$ introduced in step 1, the literal $\neg S_f(x, x_f)$ is appended to the clause.*
3. *If, after steps 1 and 2, a variable $x$ occurs in a positive, but not in a negative literal, the literal $\neg HU(x)$ is appended to the clause.*

*For a position $p$ in a clause $C$, let $\lambda(p)$ denote the corresponding position in $\lambda(C)$. For a substitution $\sigma$, let $\lambda(\sigma)$ denote the substitution obtained from $\sigma$ by replacing each assignment $x \mapsto t$ with $x \mapsto \lambda(t)$. With $\lambda^-$ we denote the inverse of $\lambda$ (that is, $\lambda^-(\lambda(\alpha)) = \alpha$ for any term, clause, position, or a substitution $\alpha$).*[2]

*For $KB$ an extensionally reduced $\mathcal{ALCHIQ}(\mathbf{D})$ knowledge base, the* function-free version *of $\Xi(KB)$, denoted with $\mathsf{FF}(KB)$, is defined as follows:*

$$\begin{aligned}
\mathsf{FF}(KB) &= \mathsf{FF}_\lambda(KB) \cup \mathsf{FF}_{Succ}(KB) \cup \mathsf{FF}_{HU}(KB) \cup \Xi(KB_\mathcal{A}) \\
\mathsf{FF}_\lambda(KB) &= \{\lambda(C) \mid C \in \mathsf{Sat}_\mathsf{R}(\Gamma_{\mathcal{TR}g})\} \\
\mathsf{FF}_{Succ}(KB) &= \{S_f(a, a_f) \mid \textit{for each } a \textit{ and } f \textit{ from } \Xi(KB)\} \\
\mathsf{FF}_{HU}(KB) &= \{HU(a) \mid \textit{for each } a \textit{ from } \Xi(KB)\} \cup \\
&\quad \{HU(a_f) \mid \textit{for each } a \textit{ and } f \textit{ from } \Xi(KB)\}
\end{aligned}$$

Note that, if a variable occurs in a positive, but not in a negative literal in a clause from $\mathsf{Sat}_\mathsf{R}(\Gamma_{\mathcal{TR}g})$, such a variable must be of the abstract sort. Therefore, it suffices to enumerate in $HU$ only the abstract part of the Herbrand universe. We now prove a lemma about the syntactic structure of clauses in $\mathsf{FF}(KB)$.

**Lemma 7.2.3.** *If $\Xi(KB_\mathcal{A})$ is $\mathsf{c}$-factored, then $\mathsf{FF}(KB)$ is $\mathsf{c}$-factored as well. Furthermore, nonground negative equality literals occur in clauses of $\mathsf{FF}(KB)$ only in disjunctions of the form $x_f \not\approx x_g \vee \neg S_f(x, x_f) \vee \neg S_g(x, x_g)$.*

*Proof.* For the first claim, observe that each clause $C$ of type 9, 12, or 13 is $\mathsf{c}$-factored, so $\lambda(C)$ is $\mathsf{c}$-factored as well. Furthermore, for a clause $C$ of type 10 and 11, a term of sort $\mathsf{c}$ occurs only in positive literals of $C$, so such terms can occur in $\lambda(C)$ only in negative literals of the form $\neg S_f(x, x_f^\mathsf{c})$. Moreover, each variable $x_f^\mathsf{c}$ occurs in exactly one such literal, so $\lambda(C)$ is $\mathsf{c}$-factored.

For the second claim, observe that $\lambda(C)$ can contain nonground equalities only if $C$ is a clause of type 5. Since such clauses only contain negative equality literals of the form $f(x) \not\approx g(x)$, $\lambda(C)$ contains $x_f \not\approx x_f \vee \neg S_f(x, x_f) \vee \neg S_g(x, x_g)$. □

We now show that eliminating function symbols does not affect satisfiability—that is, that $KB$ and $\mathsf{FF}(KB)$ are $\mathbf{D}$-equisatisfiable.

[2]Note that $\lambda$ is injective, but not surjective, so to make the definition of $\lambda^-$ is correct, we assume that $\lambda^-$ is applicable only to the range of $\lambda$.

**Lemma 7.2.4.** *KB is* **D***-unsatisfiable if and only if* $\mathsf{FF}(KB)$ *is* **D***-unsatisfiable, for an extensionally reduced* $\mathcal{ALCHIQ}(\mathbf{D})$ *knowledge base KB.*

*Proof.* Since $KB$ and $\Gamma$ are **D**-equisatisfiable by Lemma 7.2.1, the claim of this lemma can be demonstrated by showing that $\Gamma$ and $\mathsf{FF}(KB)$ are **D**-equisatisfiable. Furthermore, since $\Gamma$ and $\mathsf{FF}(KB)$ are $\mathsf{c}$-factored, by Lemma 6.1.6 it suffices to show that they are **d**-equisatisfiable.

($\Leftarrow$) If $\mathsf{FF}(KB)$ is **d**-unsatisfiable, because hyperresolution with superposition, concrete domain resolution, and splitting is sound and complete [9], a derivation of the empty clause from $\mathsf{FF}(KB)$ exists. We show that such a derivation can be transformed to a derivation of the empty clause from $\Gamma$ by sound inference rules of hyperresolution, paramodulation, instantiation, splitting, and concrete domain resolution.

Let $B$ be a branch $\mathsf{FF}(KB) = N_0, \ldots, N_n$ of a derivation from $\mathsf{FF}(KB)$ by hyperresolution with superposition, concrete domain resolution and eager splitting, where all negative literals containing a predicate other than equality are selected. We now show by induction on $n$ that a corresponding branch $B'$ in a derivation from $\Gamma$ by sound inference steps and a set of clauses $N'_m$ on $B'$ exist such that the following property (*) holds: if $C$ is a clause in $N_n$ not of the form $S_f(u, v)$ or $HU(u)$, then $N'_m$ contains the *counterpart clause* of $C$, equal to $\lambda^-(C)$. The induction base $n = 0$ is obvious, since $\mathsf{FF}(KB)$ and $\Gamma$ contain only one branch on which, other than $S_f(u, v)$ or $HU(u)$, all ground clauses are ABox clauses. Now assume that the property (*) holds for some $n$ and consider all possible inferences from premises in $N_n$ deriving a clause $C$ in $N_{n+1} = N_n \cup \{C\}$.

- For a superposition inference, observe that the clause that superposition is performed from must be ground; namely, each nonground clause is safe, so it contains negative literals, which are selected. Furthermore, because of splitting, all ground clauses are unit clauses, so superposition can be performed only from a clause of the form $s \approx t$. Consider now superposition into a ground unit clause $L$. If $L = HU(s)$, superposition is redundant, since $HU$ is instantiated for each constant occurring in $\mathsf{FF}(KB)$, so the conclusion is already contained on the branch. If $L = S_f(s, u)$ or $L = S_f(u, s)$, then the proposition obviously holds. Otherwise, clauses $s \approx t$ and $L$ are derived in at most $n$ steps on $B$, so, by the induction hypothesis, counterpart clauses $\lambda^-(s \approx t)$ and $\lambda^-(L)$ are derivable on $B'$. Thus, superposition can be performed on these clauses on $B'$ to derive the required counterpart clause. Identical arguments hold for a superposition into $\neg T(s, y^{\mathsf{c}}) \vee y^{\mathsf{c}} \not\approx_{\mathbf{D}} b^{\mathsf{c}}$. Because nonground clauses of other types contain only variables, superposition into such clauses is not possible.

- Reflexivity resolution can be applied to a ground clause $u \not\approx u$ on $B$. By the induction hypothesis, $\lambda^-(u \not\approx u)$ is then derivable on $B'$, so reflexivity resolution can be applied on $B'$ to derive the required counterpart clause. Reflexivity res-

olution is not applicable to nonground clauses, since negative literals containing the equality predicate are not selected.

- Equality factoring is not applicable to a clause on $B$, since all positive clauses on $B$ are ground unit clauses.

- Consider a hyperresolution with a main premise $C$, side premises $E_1, \ldots, E_k$, and a unifier $\sigma$, resulting in a hyperresolvent $H$. The side premises $E_i$ are not allowed to contain selected literals, so they are ground unit clauses. Furthermore, the clause $C$ is safe, and, by Lemma 7.2.3, for each literal of the form $x_f \not\approx x_g$, $C$ contains literals $\neg S_f(x, x_f)$ and $\neg S_g(x, x_g)$, respectively, which are selected. Hence, all variables in $C$ are bound, so $H$ is a ground clause. Let $\sigma'$ be the substitution obtained from $\sigma$ by including a mapping $x \mapsto \lambda^-(x\sigma)$ for each variable $x \in \mathsf{dom}(\sigma)$ not of the form $x_f$. Let us now perform an instantiation step $C' = \lambda^-(C)\sigma'$ on $B'$. Obviously, $\lambda^-(C\sigma)$ and $C'$ can differ only at a position $p$ in $C$, at which a variable of the form $x_f$ occurs. Let $p' = \lambda^-(p)$. The term in $\lambda^-(C)$ at $p'$ is of the form $f(x)$, so with $p'_x$ we denote the position of the inner $x$ in $f(x)$. In the hyperresolution inference generating $H$, the variable $x_f$ is instantiated by resolving $\neg S_f(x, x_f)$ with some ground literal $S_f(u, v)$. Hence, $C\sigma$ contains at $p$ the term $v$, whereas $C'$ contains at $p'$ the term $f(u)$, and $\lambda^-(v) \neq f(u)$. We show that all such discrepancies can be eliminated with sound inferences on $B'$. The literal $S_f(u, v)$ is obtained on $B$ from some $S_f(a, a_f)$ by $n$ or less superposition inference steps. Let us with $\Delta_1$ ($\Delta_2$) denote the sequence of ground unit equalities applied to the first (second) argument of $S_f(a, a_f)$. All $s_i \approx t_i$ from $\Delta_1$ and $\Delta_2$ are derivable on $B$ in $n$ steps or less, so corresponding equalities $\lambda^-(s_i \approx t_i)$ are derivable on $B'$ by the induction hypothesis; we denote these sequences with $\Delta'_1$ and $\Delta'_2$. We now perform superposition with equalities from $\Delta'_1$ into $C'$ at $p'_x$ in the reverse order. After this, $p'_x$ contains the constant $a$, and $p'$ contains the term $f(a)$. Hence, we can apply superposition with equalities from $\Delta'_2$ at $p'$ in the original order. After this is done, each position $p'$ contains the term $\lambda^-(v)$. Let $C''$ denote the result of removing discrepancies at all positions; obviously, $C'' = \lambda^-(C\sigma)$. All side premises $E_i$ are derivable in $n$ steps or less on $B$, so, if $E_i$ is not of the form $S_f(u, v)$ or $HU(u)$, $\lambda^-(E_i)$ is derivable on $B'$. We hyperresolve these side premises with $C''$ to obtain $H'$. Obviously, $H' = \lambda^-(H)$, so the counterpart clause is derivable on $B'$.

- Since nonground clauses contain selected literals, and all concrete domain literals are positive, concrete domain resolution can be applied only to a set of positive ground clauses $C_i$. By the induction hypothesis, all $\lambda^-(C_i)$ are derivable on $B'$. Hence, concrete domain resolution can be applied on $B'$ in the same way as on $B$, so the counterpart clause is derivable on $B'$.

- Since all ground clauses on $B$ are unit clauses, and no clause in $\mathsf{FF}(KB)$ contains a positive literal with $S_f$ or $HU$ predicates, a ground nonunit clause $C$ generated

by an inference on $B$ cannot contain $S_f(u, v)$ and $HU(a)$ literals. Hence, if $C$ is of length $k$ and causes $B$ to be split into $k$ sub-branches, then $\lambda^-(C)$ is of length $k$ and $B'$ can be split into $k$ sub-branches, each of them satisfying (*).

Hence, if there is a derivation of the empty clause on all branches from $\mathsf{FF}(KB)$, then there is a derivation of the empty clause on all branches from $\Gamma$ as well.

($\Rightarrow$) If $\Gamma$ is **d**-unsatisfiable, since $\mathcal{BS}_{DL}^{\mathbf{D},+}$ is sound and complete, a derivation of the empty clause from $\Gamma$ exists. We show that such a derivation can be reduced to a derivation of the empty clause in $\mathsf{FF}(KB)$ by sound inference rules.

Let $B'$ be a derivation $\Gamma = N'_0, \ldots, N'_n$ by $\mathcal{BS}_{DL}^{\mathbf{D},+}$. We show by induction on $n$ that there exists a corresponding derivation $B$ of the form $\mathsf{FF}(KB) = N_0, \ldots, N_m$ by sound inference steps such that the following property (**) holds: if $C'$ is a clause in $N'_n$, then $N_m$ contains the counterpart clause $C = \lambda(C')$. The induction base $n = 0$ is trivial. Let us now assume that (**) holds for some $n$, and let us consider possible inferences deriving $N'_{n+1} = N'_n \cup \{C'\}$, where the clause $C'$ is derived from premises $P'_i \in N'_n, 1 \leq i \leq k$. By the induction hypothesis, we know that there is a derivation $B$ from $\mathsf{FF}(KB)$ with a clause set $N_m$, containing the counterpart clauses of each $P'_i$, denoted with $P_i, 1 \leq i \leq k$. Let $\sigma'$ be the unifier of the inference. By Corollary 6.2.2, $\sigma'$ is ground and contains only assignments of the form $x_i \mapsto a$ or $x_i \mapsto f(a)$. Let $\sigma = \lambda(\sigma')$. Since all $P_i$ are derivable by the induction hypothesis, we can instantiate each $P_i$ into $P_i\sigma$. Obviously, apart from the literals involving $S_f$ and $HU$, the only difference between $P_i\sigma$ and $\lambda(P'_i\sigma')$ may be that the latter contains a term $f(a)$ at position $p$, whereas the former contains $x_f$ at $\lambda(p)$. But then $P_i\sigma$ contains a literal $\neg S_f(a, x_f)$, which can be resolved with $S_f(a, a_f)$ to produce $a_f$ at $\lambda(p)$. All such differences can be removed iteratively, and the remaining ground literals involving $HU$ can be resolved away. Hence, each $\lambda(P'_i\sigma')$ is derivable from premises in $N_m$.

Observe that in all literals of the form $f(a)$, the inner term is marked. Hence, superposition inferences are possible only on the outer position of such terms, which correspond via $\lambda$ to $a_f$. Therefore, regardless of the inference type, $C = \lambda(C')$ can be derived from $\lambda(P'_i\sigma')$ by the same inference on the corresponding literals.

The result of a superposition inference in $B'$ may be a clause $C'$ containing a literal $R([a], [f(a)])$, which is decomposed into a clause $C'_1$ of type 8 and a clause $C'_2$ of type 3. However, since $\mathsf{gen}(KB) \subseteq \Gamma$, we have $C'_2 \in \Gamma$, so the conclusion $C'$ should only be replaced with the conclusion $C'_1$. The decomposition inference rule can obviously be applied on $B$ as well to produce a counterpart clause $C_1 = \lambda(C'_1)$. Since $\lambda(C'_2) \in N_m$, this inference is sound by Lemma 5.4.2, so the property (**) holds.

Now it is obvious that, if there is a derivation of the empty clause from $\Gamma$, then there is a derivation of the empty clause from $\mathsf{FF}(KB)$ as well. □

Lemma 7.2.4 also implies that $KB \models \alpha$ if and only if $\mathsf{FF}(KB) \models \alpha$, where $\alpha$ is of the form $(\neg)A(a)$ or $(\neg)R(a, b)$, for $A$ an atomic concept and $R$ a simple role. The proof also reveals that, in checking **D**-satisfiability of $\mathsf{FF}(KB)$, it is not necessary to perform a superposition inference into a literal of the form $HU(a)$.

It is interesting to consider the role of the new constants $a_f$ in $\mathsf{FF}(KB)$. As discussed in Section 4.6, our transformation does not preserve the semantics of $KB$—the models of $\mathsf{FF}(KB)$ and $KB$ are, in general, not related, and they coincide only on positive ground facts. Therefore, a constant $a_f$ can be intuitively best understood as a proof-theoretic means used to simulate the term $f(a)$ from a refutation by basic superposition. We do not provide any results about a deeper correspondence between $a_f$ and unnamed individuals in a model of $KB$.

## 7.3 Removing Irrelevant Clauses

The saturation of $\Gamma_{\mathcal{TR}g}$ derives new clauses that enable the reduction to $\mathsf{FF}(KB)$. However, the same process introduces many clauses that are not necessary. Consider, for example, the knowledge base $KB = \{A \sqsubseteq C, C \sqsubseteq B\}$. If the precedence of the predicate symbols is $C > B > A$, the saturation process derives $\neg A(x) \vee B(x)$; however, this clause is entailed by the premises, and it can be deleted. Hence, we present an optimization, by which we reduce the number of rules in the resulting disjunctive datalog program.

**Definition 7.3.1.** *For $N \subseteq \mathsf{FF}(KB)$, let $C \in N$ be a clause such that $\lambda^-(C)$ is derived in the saturation of $\Gamma_{\mathcal{TR}g}$ from premises $P_i$, $1 \leq i \leq k$, by an inference with a substitution $\sigma$. Then, $C$ is* irrelevant w.r.t. $N$ *if the following conditions hold:*

- *$\lambda^-(C)$ is not derived by the decomposition rule;*
- *For each premise $P_i$, $\lambda(P_i) \in N$;*
- *Each variable occurring in $\lambda(P_i\sigma)$ occurs in $C$.*

Relevant *is the opposite of irrelevant. Let $C_1, C_2, \ldots, C_n$ be a sequence of clauses from $\mathsf{FF}(KB)$ such that $\lambda^-(C_n), \ldots, \lambda^-(C_2), \lambda^-(C_1)$ is the order in which the clauses are derived in the saturation of $\Gamma_{\mathcal{TR}g}$. Let $\mathsf{FF}(KB) = N_0, N_1, \ldots, N_n$ be a sequence of clause sets such that $N_i = N_{i-1}$ if $C_i$ is relevant w.r.t. $N_{i-1}$, and $N_i = N_{i-1} \setminus \{C_i\}$ if $C_i$ is irrelevant w.r.t. $N_{i-1}$, for $1 \leq i \leq n$. Then, $\mathsf{FF}_\mathsf{R}(KB) = N_n$ is called the* relevant subset *of $\mathsf{FF}(KB)$.*

Removing irrelevant clauses preserves satisfiability, as demonstrated by the following lemma.

**Lemma 7.3.2.** *$\mathsf{FF}_\mathsf{R}(KB)$ is $\mathbf{D}$-unsatisfiable if and only if $\mathsf{FF}(KB)$ is $\mathbf{D}$-unsatisfiable.*

*Proof.* Let $N$ be a (not necessarily proper) subset of $\mathsf{FF}(KB)$. Furthermore, let $C \in N$ be an irrelevant clause w.r.t. $N$, where $\lambda^-(C)$ is derived in the saturation of $\Gamma_{\mathcal{TR}g}$ from premises $P_i$ by an inference $\xi$ with a substitution $\sigma$. Finally, let $\lambda(P_i) \in N$, $i \leq i \leq k$. We now show the following property (*): $N$ is $\mathbf{D}$-unsatisfiable if and only if $N \setminus \{C\}$ is $\mathbf{D}$-unsatisfiable. The ($\Leftarrow$) direction is trivial, since $N \setminus \{C\} \subset N$. For the ($\Rightarrow$)

direction, by Herbrand's theorem, $N$ is **D**-unsatisfiable if and only if some finite set $M$ of ground instances of $N$ is **D**-unsatisfiable. For such $M$, we construct the set of ground clauses $M'$ in the following way:

- For each $D \in M$ such that $D$ is not a ground instance of $C$, let $D \in M'$;
- For each $D \in M$ such that $D$ is a ground instance of $C$ with substitution $\tau$, let $\lambda(P_i)\lambda(\sigma)\tau \in M'$, $1 \leq i \leq k$.

Let $\tau$ be a ground substitution such that $D = C\tau$. Clauses $P_i$ can be of type 1, 2, 3, 5, 7, or 9–13, so $\sigma$ can contain only mappings of the form $x \mapsto x'$, $x \mapsto f(x')$, or $y_i \mapsto f(x')$. The sets of variables in $\lambda(P_i\sigma)$ and $\lambda(P_i)\lambda(\sigma)$ obviously coincide. Since $C$ is irrelevant, each variable from each $\lambda(P_i\sigma)$ occurs in $C$ as well, so $\tau$ instantiates all variables in all $\lambda(P_i)\lambda(\sigma)$. Therefore, each $\lambda(P_i)\lambda(\sigma)\tau$ is a ground instance of a clause $\lambda(P_i)$ in $N \setminus \{C\}$. Furthermore, $\lambda(P_i)\lambda(\sigma)\tau \subseteq \lambda(P_i\sigma)\tau$. If the inclusion is strict, then the latter clause contains literals of the form $\neg S_f(a, b)$ that do not occur in the former one, because $\sigma$ instantiates a variable from $P_i$ to a term $f(x')$ originating from some premise $P_j$. But then, $\lambda(P_j)$ contains the literal $\neg S_f(x', x'_f)$, so $\lambda(P_j)\lambda(\sigma)\tau$ contains $\neg S_f(a, b)$. Therefore, all $\lambda(P_i)\lambda(\sigma)\tau$ can participate in a ground inference corresponding to $\xi$, so $D$ can be derived from $M'$. Hence, if $M$ is **D**-unsatisfiable, then $M'$ is **D**-unsatisfiable as well. Since $M'$ is a finite **D**-unsatisfiable set of ground instances of $N \setminus \{C\}$, the latter is **D**-unsatisfiable by Herbrand's theorem, so the property (*) holds.

Consider the sequence of clause sets $\mathsf{FF}(KB) = N_0, N_1, \ldots, N_n = \mathsf{FF}_\mathsf{R}(KB)$ from Definition 7.3.1. For each $N_i = N_{i-1} \setminus \{C_i\}$, $i \geq 1$, the preconditions of property (*) are fulfilled, so by (*), $N_i$ is **D**-satisfiable if and only if $N_{i-1}$ is **D**-satisfiable. The claim of the lemma now follows by a straightforward induction on $i$. □

## 7.4 Reduction to Disjunctive Datalog

Reduction of an $\mathcal{ALCHIQ}(\mathbf{D})$ knowledge base $KB$ to a disjunctive datalog program is now easy: for each clause from $\mathsf{FF}_\mathsf{R}(KB)$, simply move all positive literals into the rule head, and all negative literals into the rule body.

**Definition 7.4.1.** *For an extensionally reduced $\mathcal{ALCHIQ}(\mathbf{D})$ knowledge base $KB$, $\mathsf{DD}(KB)$ is the disjunctive datalog program that contains the rule*

$$A_1 \vee \ldots \vee A_n \leftarrow B_1, \ldots, B_m$$

*for each clause $A_1 \vee \ldots \vee A_n \vee \neg B_1 \vee \ldots \vee \neg B_m$ from $\mathsf{FF}_\mathsf{R}(KB)$. If $KB$ is not extensionally reduced, then $\mathsf{DD}(KB) = \mathsf{DD}(KB')$, where $KB'$ is an extensionally reduced knowledge base obtained from $KB$ as explained in Section 3.1.*

**Theorem 7.4.2.** *Let $KB$ be an $\mathcal{ALCHIQ}(\mathbf{D})$ knowledge base, defined over a concrete domain $\mathbf{D}$ such that $\mathbf{D}$-satisfiability of finite conjunctions over $\Phi_{\mathbf{D}}$ can be decided in deterministic exponential time. Then, the following claims hold:*

1. *$KB$ is $\mathbf{D}$-unsatisfiable if and only if $\mathsf{DD}(KB)$ is $\mathbf{D}$-unsatisfiable.*
2. *$KB \models_{\mathbf{D}} \alpha$ if and only if $\mathsf{DD}(KB) \models_c \alpha$, where $\alpha$ is of the form $A(a)$ or $R(a, b)$, and $A$ is an atomic concept.*
3. *$KB \models_{\mathbf{D}} C(a)$ for a nonatomic concept $C$ if and only if, for $Q$ a new atomic concept, $\mathsf{DD}(KB \cup \{C \sqsubseteq Q\}) \models_c Q(a)$.*
4. *The number of literals in each rule in $\mathsf{DD}(KB)$ is at most polynomial, the number of rules in $\mathsf{DD}(KB)$ is at most exponential, and $\mathsf{DD}(KB)$ can be computed in exponential time in $|KB|$, assuming a bound on the arity of the concrete domain predicates and for unary coding of numbers in input.*

*Proof.* The first claim follows directly from Lemma 7.3.2. The second claim follows from the first one, since $\mathsf{DD}(KB \cup \{\neg\alpha\}) = \mathsf{DD}(KB) \cup \{\neg\alpha\}$ is $\mathbf{D}$-unsatisfiable if and only if $\mathsf{DD}(KB) \models_c \alpha$. Furthermore, $KB \models C(a)$ if and only if $KB \cup \{\neg C(a)\}$ is $\mathbf{D}$-unsatisfiable, which is the case if and only if $KB \cup \{\neg Q(a), C \sqsubseteq Q\}$ is $\mathbf{D}$-unsatisfiable. Now, because $Q$ is atomic, the third claim follows from the second one.

By Lemma 5.3.9, for each clause $C \in \mathsf{Sat}(\Gamma_{\mathcal{TR}g})$, the number of literals in $C$ is at most polynomial in $|KB|$, and $|\mathsf{Sat}(\Gamma_{\mathcal{TR}g})|$ is at most exponential in $|KB|$. It is easy to see that the application of $\lambda$ to $C$ can be performed in time polynomial in the number of terms and literals in $C$. The number of constants $a_f$ added to $\mathsf{DD}(KB)$ is equal to $c \cdot f$, where $c$ is the number of constants, and $f$ the number of function symbols in the signature of $\Xi(KB)$. If numbers are coded in unary, both $c$ and $f$ are polynomial in $|KB|$, so the number of constants $a_f$ is also polynomial in $|KB|$. By Theorem 6.2.5, $\mathsf{Sat}(\Gamma_{\mathcal{TR}g})$ can be computed in time that is at most exponential in $|KB|$. □

We point out that basic superposition is crucial for the correctness of the reduction. Namely, in basic superposition, superposition into Skolem function symbols is redundant, which allows encoding ground terms of the form $f(a)$ using new constants.

## 7.5 Equality Reasoning in $\mathsf{DD}(KB)$

The program $\mathsf{DD}(KB)$ can contain the equality predicate in the rule heads, which is not allowed in usual definitions of disjunctive datalog, such as [42]. Hence, to answer queries over $\mathsf{DD}(KB)$, we apply a well-known transformation from [46], which allows us to treat $\approx$ as an ordinary predicate. The effects of this transformation were discussed in more detail in Subsection 4.8.5.

For a disjunctive datalog program $P$, with $P_{\approx}$ we denote the program consisting of the rules (7.1)–(7.4), where (7.1) is instantiated for each individual $a$ occurring in

$P$, and (7.4) is instantiated for each predicate symbol $R$ occurring in $P$. (Note that it is not necessary to instantiate (7.4) for $R = \approx$, since such a rule logically follows from symmetry and transitivity.) From [46, 26], it is well known that, for $\varphi$ a first-order formula, $P \models \varphi$ (where $\models$ denotes the usual first-order entailment w.r.t. an equational theory) if and only if $P \cup P_{\approx} \models_f \varphi$ (where $\models_f$ denotes first-order entailment where equality is not given any special meaning). In other words, appending $P_{\approx}$ to $P$ allows us to treat $\approx$ as an ordinary predicate, and to use any existing reasoning technique for disjunctive datalog without equality.

$$a \approx a \tag{7.1}$$

$$x \approx y \leftarrow y \approx x \tag{7.2}$$

$$x \approx z \leftarrow x \approx y, y \approx z \tag{7.3}$$

$$R(x_1, \ldots, y_i, \ldots, x_n) \leftarrow R(x_1, \ldots, x_i, \ldots, x_n), x_i \approx y_i \tag{7.4}$$

The results in [46, 26] consider only first-order logic without concrete domains. However, by Definition 3.2.2, only concrete equality $\approx_{\mathbf{D}}$ is allowed for concrete terms. Since $\approx_{\mathbf{D}}$ is treated like any other concrete predicate, by instantiating (7.4) in $P_{\approx}$ only for positions of predicates that are of sort other than $\mathsf{c}$, the encoding is sound and complete even if $P$ contains concrete predicates.

Observe also that (7.4) is instantiated at most for each predicate $R$ and each position $i$ in $R$. Since the number of predicates and the number of positions are both linear in $|P|$, this does not increase the complexity of reasoning.

The presented approach to handling equality may prove inefficient in practice, since the rules from $P_{\approx}$ can be used to derive the same conclusion multiple times. Therefore, we present an optimization that allows us to omit certain instances of (7.4).

**Theorem 7.5.1.** *For a disjunctive datalog program $P$, let $P^F_{\approx}$ be the subset of $P_{\approx}$, where* (7.4) *is instantiated only for those predicates $R$ and positions $i$, for which $R$ occurs either in a head or a body atom $R(\mathbf{t})$ of some rule in $P$, and $\mathbf{t}$ contains a constant at position $i$. Then, the following claims hold:*

1. *$P \cup P_{\approx}$ is satisfiable if and only if $P \cup P^F_{\approx}$ is satisfiable.*
2. *$P \cup P_{\approx} \models_f Q(\mathbf{a})$ if and only if $P \cup P^F_{\approx} \cup P^Q_{\approx} \models_f Q(\mathbf{a})$ for a ground atom $Q(\mathbf{a})$, where $P^Q_{\approx}$ is obtained by instantiating* (7.4) *for all positions in $Q$.*

*Proof.* (Claim 1.) Since $P^F_{\approx} \subseteq P_{\approx}$, the ($\Rightarrow$) direction is trivial. For the ($\Leftarrow$) direction, let $I$ be a Herbrand model of $P \cup P^F_{\approx}$. Because $P^F_{\approx}$ contains the axioms (7.1), (7.2), and (7.3), the relation $\approx$ is an equivalence relation. Let us choose a distinct object for each equivalence class of $\approx$, and let $\widetilde{a}$ denote the object chosen for the equivalence class that $a$ belongs to. For $\alpha$ a substitution, atom, literal, or a rule, with $\widetilde{\alpha}$ we denote the result of replacing each $a$ in $\alpha$ with $\widetilde{a}$. Finally, let $I_{\approx}$ be the smallest Herbrand interpretation such that $I \subseteq I_{\approx}$ and $R(\widetilde{a}_1, \ldots, \widetilde{a}_n) \in I$ implies $R(a_1, \ldots, a_n) \in I$ for all distinct constants $a_i$. We now show that $I_{\approx}$ is a model of $P \cup P_{\approx}$.

The interpretation $I_\approx$ obviously satisfies all rules from $P_\approx$. Let $C$ be a rule from $P$ (for the purpose of this proof, we consider a rule to be equivalent to a clause) and $\tau$ a substitution such that $C\tau$ is ground. Because $I \models_f P \cup P^F_\approx$, we have $I \models_f C\widetilde{\tau}$. Let $L \in C$ be a literal such that $I \models_f L\widetilde{\tau}$. Because the rule (7.4) is instantiated in $P^F_\approx$ for positions at which $L$ contains constants, $I \models_f \widetilde{L\widetilde{\tau}}$. By definition, $I$ and $I_\approx$ coincide for ground atoms such that $A = \widetilde{A}$; hence, $I_\approx \models_f \widetilde{L\widetilde{\tau}}$. Finally, $I_\approx$ is an interpretation in which $\approx$ is a congruence relation, so $I_\approx \models_f L\tau$, and therefore $I_\approx \models_f C\tau$. Because this holds for all substitutions $\tau$ and rules $C$, we have $I_\approx \models_f P$.

(Claim 2.) Observe that $P \cup P_\approx \models_f Q(\mathbf{a})$ if and only if $P \cup P_\approx \cup \{\neg Q(\mathbf{a})\} = P'$ is unsatisfiable. Since $Q$ occurs in $\neg Q(\mathbf{a})$ in a negative literal, we instantiate (7.4) for $Q$, which yields the set of rules $P^Q_\approx$. By the first claim of the theorem, $P'$ is unsatisfiable if and only if $P \cup P^F_\approx \cup P^Q_\approx\{\neg Q(\mathbf{a})\}$ is unsatisfiable, which is the case if and only if $P \cup P^F_\approx \cup P^Q_\approx \models_f Q(\mathbf{a})$. □

Usually, in (disjunctive) datalog one distinguishes *extensional predicates*, which occur only in facts and rule bodies, from *intensional predicates*, which occur in the heads of rules with a nonempty body. If the rules of $P$ do not contain constants, Theorem 7.5.1 states that, to compute the set of positive ground consequences of $P$ w.r.t. an equational theory, it suffices to instantiate (7.4) only for extensional predicates and the query predicate.

Another optimization is applicable to an atom $A$ of the form $x \approx t$ occurring in the body of a rule $r \in P$. Namely, for a first-order formula $\varphi$, a variable $x$, and a term $t$ that does not contain $x$ as a proper subterm, the formulae $\forall x : (x \approx t \rightarrow \varphi)$ and $\varphi\{x \mapsto t\}$ are equisatisfiable [99]. Hence, we can delete $A$ from the body of $r$, and replace all occurrences of $x$ with $t$; if the resulting rule contains an unsafe variable $y$, we append the literal $HU(y)$. (As explained in Section 7.2, $HU$ contains all constants occurring in $P$.) Furthermore, a literal of the form $a \approx a$ can simply be deleted from the body of $r$, since such a literal is always true. This optimization has an important side-effect: it allows us to omit all instances of (7.1) from $P^F_\approx$. Namely, the reflexivity axioms $a \approx a$ can only participate in an inference with a body literal $x \approx y$ or $a \approx a$; however, the literals of the latter form are eagerly eliminated as explained previously.

## 7.6 Answering Queries in $\mathsf{DD}(KB)$

Currently, the state-of-the-art technique for reasoning in disjunctive datalog is *intelligent grounding* [43]. It is based on model building, which is performed by generating a ground instantiation of the program rules, generating candidate models, and eliminating those candidates that do not satisfy the ground rules. In order to avoid generating the entire grounding of the program, carefully designed heuristics are applied to generate the subset of the ground rules that have exactly the same set of the stable models as the original program. This technique has been successfully implemented in the disjunctive datalog engine DLV [43].

Model building is of central interest in many applications of disjunctive datalog. For example, disjunctive datalog has been successfully applied to planning problems, where plans are decoded from models. Query answering is easily reduced to model building: $A$ is not a certain answer if and only if there is a model not containing $A$. In our view, the main drawback of query answering by model building is that such an approach provides answering of ground queries only; nonground queries are typically answered by considering all ground instances of the query. Furthermore, the algorithm computes the grounded program, which can be large even for small nonground programs.

Note that the models of $\mathsf{DD}(KB)$ are of no interest, as they do not reflect the structure of the models of $KB$ (see Section 4.6 for an example). Hence, we propose to answer nonground queries in $\mathsf{DD}(KB)$ by ordered hyperresolution, which may be viewed as an extension of the fixpoint computation of plain datalog. This algorithm computes the set of all answers to a nonground query in one pass, and does not consider each ground instance separately. Furthermore, the algorithm does not compute the program grounding, but works with the nonground program directly. Finally, the algorithm exhibits optimal worst-case complexity. A similar technique was presented in [25]; however, the algorithm presented there does not specify whether application of redundancy elimination techniques is allowed during hyperresolution. Both techniques are actually extensions of the *answer literal* technique, first proposed in [56].

Roughly speaking, we saturate $\mathsf{DD}(KB) \cup \mathsf{DD}(KB)_{\approx}$ by $\mathcal{R}^{\mathbf{D}}$, with all negative literals selected, under an ordering in which all ground query literals are smallest. Additionally, we require that the query predicate does not occur in the body of any rule. Under these assumptions, the saturated set of clauses contains the cautious consequences w.r.t. the query predicate as unit clauses. Because the ordering is total, in each ground disjunction there is exactly one maximal literal, so the seminaïve bottom-up saturation [112] or various join order optimizations can easily be adapted to the disjunctive case.

**Definition 7.6.1.** *For a predicate symbol $Q$, let $\mathcal{R}^{\mathbf{D}}_Q$ denote the $\mathcal{R}^{\mathbf{D}}$ calculus parameterized as follows:*

- *All negative literals are selected.*
- *All ground atoms of the form $Q(\mathbf{a})$ are smallest in the ordering $\succ$.*

A simple ordering compatible with $\mathcal{R}^{\mathbf{D}}_Q$ can be obtained in the following way. Let $\succ_P$ and $\succ_C$ be orderings on predicate and constant symbols, respectively, such that the query predicate $Q$ is the smallest element in $\succ_P$. Now $A(a_1, \ldots, a_n) \succ_Q B(b_1, \ldots, b_m)$ if and only if $A \succ_P B$, or $A = B$ and an index $i$ exists such that $a_j = b_j$ for $j < i$ and $a_i \succ_C b_i$. It is easy to see that the ordering $\succ_Q$ is compatible with Definition 7.6.1. Since in $\mathcal{R}^{\mathbf{D}}$ all negative literals are selected, the definition of $\succ_Q$ on negative literals is not important.

**Lemma 7.6.2.** *Let $P$ be a positive satisfiable disjunctive datalog program, $Q$ a predicate not occurring in the body of any rule in $P$, and $Q(\mathbf{a})$ a ground literal not containing*

*concrete terms. Then $P \models_c Q(\mathbf{a})$ if and only if $Q(\mathbf{a}) \in N$, where $N$ is the set of clauses obtained by saturating a* c*-factor of $P$ under $\mathcal{R}_Q^{\mathbf{D}}$ up to redundancy.*

*Proof.* Let $P'$ be a c-factor of $P$. Since $\mathcal{R}_Q^{\mathbf{D}}$ is sound and complete, $P \models_c Q(\mathbf{a})$ if and only if the set of clauses $N'$, obtained by saturating $P' \cup \{\neg Q(\mathbf{a})\}$ by $\mathcal{R}_Q^{\mathbf{D}}$ up to redundancy, contains the empty clause. Note that, since all clauses in $P'$ are safe and all negative literals are selected, all hyperresolvents are positive ground clauses.

Since $P$ is satisfiable, $N'$ contains the empty clause if and only if a hyperresolution step with $\neg Q(\mathbf{a})$ is performed in the saturation by $\mathcal{R}_Q^{\mathbf{D}}$. Since the literals containing $Q$ are smallest in the ordering of $\mathcal{R}_Q^{\mathbf{D}}$, a positive literal $Q(\mathbf{a})$ can be maximal only in a clause $C = Q(\mathbf{a}) \vee D$, where $D$ contains only literals with the $Q$ predicate. Since $\neg Q(\mathbf{a})$ is the only clause where $Q$ occurs negatively, if $D$ is not empty, no literal from $D$ can be eliminated by a subsequent hyperresolution inference. Hence, the empty clause can be derived from such $C$ if and only if $D$ is empty, but then $Q(\mathbf{a}) \in N$. □

Assuming that $Q$ does not occur in the body of any rule in $P$ does not reduce the generality of the approach, as one can always add a new rule of the form $A_Q(\mathbf{x}) \leftarrow Q(\mathbf{x})$ to satisfy the lemma conditions. Also, note that redundancy elimination techniques can freely be applied in the saturation. We now consider the complexity of answering queries in $\mathsf{DD}(KB)$ by $\mathcal{R}_Q^{\mathbf{D}}$.

**Theorem 7.6.3.** *Let $KB$ be an $\mathcal{ALCHIQ}(\mathbf{D})$ knowledge base, defined over a concrete domain $\mathbf{D}$ such that $\mathbf{D}$-satisfiability of finite conjunctions over $\Phi_{\mathbf{D}}$ can be decided in deterministic exponential time. Then, the set of all $\alpha$ such that $\mathsf{DD}(KB) \models_c \alpha$, for $\alpha$ the form $C(a)$ or $R(a, b)$ with $R$ an abstract role, can be computed by saturating a* c*-factor of $\mathsf{DD}(KB) \cup \mathsf{DD}(KB)_{\approx}$ with $\mathcal{R}_Q^{\mathbf{D}}$ in time exponential in $|KB|$, assuming a bound on the arity of concrete predicates and unary coding of numbers in input.*

*Proof.* Let $P$ be the c-factor of $\mathsf{DD}(KB) \cup \mathsf{DD}(KB)_{\approx}$. Since the number of predicates in $\mathsf{DD}(KB)$ is linear in $|KB|$, the number of rules in $\mathsf{DD}(KB)_{\approx}$ is linear in $|KB|$. In a way similar as in the proof of Lemma 5.3.9, it is easy to see that the maximal length of each ground clause obtained in the saturation of $P$ by $\mathcal{R}_Q^{\mathbf{D}}$ is polynomial in $|KB|$, so the number of ground clauses is exponential in $|KB|$. Furthermore, in each application of the hyperresolution inference to some rule $r$, one selects a ground clause for each body literal of $r$. Since the length of the rules is polynomial in $|KB|$, there are exponentially many selections of ground clauses, giving rise to exponentially many hyperresolution inferences. Hence, the saturation of $P$ can be performed in time exponential in $|KB|$ and, by Lemma 7.6.2, it computes all certain consequences of $P$ regarding the query predicate, so the claim of the theorem follows. □

We briefly discuss the restriction of Theorem 7.6.3 that $R$ is an abstract role. Consider the knowledge base $KB$ containing only the axiom $\exists T.{=_5}(a)$. Obviously, $KB \models T(a, 5)$, where, by convention from Subsection 3.2.1, $T(a, 5)$ is a shortcut for $T(a, a_5) \wedge {=_5}(a_5)$. The disjunctive datalog program $\mathsf{DD}(KB)$ consists of these rules:

$$T(x, x_f) \leftarrow Q_1(x), S_f(x, x_f) \tag{7.5}$$

$$=_5(x_f) \leftarrow Q_1(x), S_f(x, x_f) \tag{7.6}$$

$$Q_1(a) \tag{7.7}$$

$$S_f(a, a_f) \tag{7.8}$$

The translation into disjunctive datalog does not change the set of entailed facts: $\mathsf{DD}(KB) \models_c T(a, 5)$ as well. The latter we demonstrate proof-theoretically by showing that $P = \mathsf{DD}(KB) \cup \{\neg T(a, 5)\}$ is unsatisfiable. To translate $\neg T(a, 5)$ into clauses, we apply c-factoring and obtain (7.9):

$$x \not\approx_{\mathbf{D}} 5 \leftarrow T(a, x) \tag{7.9}$$

We now saturate $P$ under $\mathcal{R}_Q^{\mathbf{D}}$. Observe that (7.13) is derived by concrete domain resolution, since the set $S = \{a_f \not\approx_{\mathbf{D}} 5, =_5(a_f)\}$ is a $\mathbf{D}$-constraint.

| | | |
|---|---|---|
| (7.10) | $T(a, a_f)$ | R(7.5;7.7;7.8) |
| (7.11) | $a_f \not\approx_{\mathbf{D}} 5$ | R(7.9;7.10) |
| (7.12) | $=_5(a_f)$ | R(7.6;7.7;7.8) |
| (7.13) | $\square$ | D(7.11;7.12) |

However, $\mathsf{DD}(KB)$ does not contain the constant 5; this constant was added in the refutation only by assuming the negation of the goal $T(a, 5)$. Hence, the fact $T(a, 5)$ cannot be derived by saturating $\mathsf{DD}(KB)$ by $\mathcal{R}_Q^{\mathbf{D}}$. To summarize, we can freely use $\mathsf{DD}(KB)$ to answer queries by refutation, but we cannot use $\mathcal{R}_Q^{\mathbf{D}}$ to answer queries regarding concrete roles or concrete literals.

## 7.7 Example

We now show how to translate a subset of $KB_1 \cup KB_2 \cup KB_3 \cup KB_4$ into disjunctive datalog ($KB_1$ and $KB_2$ are given in Subsection 3.1.1, while $KB_3$ and $KB_4$ are given in Subsection 3.2.3). Namely, the axioms involving the role *contains* (in particular the transitivity axiom) generate many closures that participate in many inferences, but do not contribute significantly to understanding the algorithm. Hence, to make the example easier to follow, let $KB_5$ contain the axioms (3.1)–(3.9), (3.12)–(3.14), (3.16), (3.18)–(3.21), and the following axiom (7.14):

$$\exists \mathit{hasVD}.\mathit{Adpt3DAcc} \sqsubseteq Q_1 \tag{7.14}$$

The axiom (7.14) introduces a new name for the concept $\exists \mathit{hasVD}.\mathit{Adpt3DAcc}$. Hence, to show that $KB_5 \models \exists \mathit{hasVD}.\mathit{Adpt3DAcc}(\mathit{pc}_1)$, we check whether $KB_5 \models \exists Q_1(\mathit{pc}_1)$.

**Translation into Closures.** Translating $KB_5$ into closures is explained in Subsections 5.5 and 6.3. We recapitulate all closures from $\Xi(KB_5)$, but we renumber the predicates in $\Xi(KB_5)$ introduced by structural transformation to use consecutive indices. We also reorder the closures to make the order of applying inferences easier to follow. The following closures correspond to axioms from $KB_1$ and $KB_2$:

$$\neg PC(x) \vee hasAdpt(x, f(x)) \tag{7.15}$$
$$\neg GrWS(x) \vee has3DAcc(x, g(x)) \tag{7.16}$$
$$\neg hasAdpt(x, y) \vee hasVD(x, y) \tag{7.17}$$
$$\neg has3DAcc(x, y) \vee hasVD(x, y) \tag{7.18}$$
$$\neg hasAdpt(x, y) \vee Adpt(y) \tag{7.19}$$
$$\neg has3DAcc(x, y) \vee 3DAcc(y) \tag{7.20}$$
$$\neg GaPC(x) \vee \neg hasVD(x, y_1) \vee \neg hasVD(x, y_2) \vee y_1 \approx y_2 \vee \neg VD(y_1) \vee \neg VD(y_2) \tag{7.21}$$
$$\neg Adpt(x) \vee VD(x) \tag{7.22}$$
$$\neg 3DAcc(x) \vee VD(x) \tag{7.23}$$
$$\neg Adpt(x) \vee \neg 3DAcc(x) \vee Adpt3DAcc(x) \tag{7.24}$$
$$\neg GrWS(x) \vee PC(x) \tag{7.25}$$
$$\neg GaPC(x) \vee PC(x) \tag{7.26}$$
$$\neg GaPC(x) \vee GrWS(x) \tag{7.27}$$
$$GaPC(pc_1) \tag{7.28}$$

The following closure corresponds to the axiom (7.14):

$$Q_1(x) \vee \neg hasVD(x, y) \vee \neg Adpt3DAcc(y) \tag{7.29}$$

The following closures correspond to axioms from $KB_3$ and $KB_4$:

$$\neg Q_2(x) \vee price(x, h(x)) \tag{7.30}$$
$$\neg Q_2(x) \vee \geq_{2000}(h(x)) \tag{7.31}$$
$$\neg Q_3(x) \vee price(x, i(x)) \tag{7.32}$$
$$\neg Q_3(x) \vee \leq_{1500}(i(x)) \tag{7.33}$$
$$\neg price(x, y_1) \vee \neg price(x, y_2) \vee y_1 \approx_{\mathbf{D}} y_2 \tag{7.34}$$
$$\neg HighEnd(x) \vee GrWS(x) \vee Q_2(x) \tag{7.35}$$
$$\neg Q_2(x) \vee PC(x) \tag{7.36}$$
$$Q_3(pc_2) \tag{7.37}$$
$$HighEnd(pc_2) \tag{7.38}$$

Finally, we introduce the closures from $\mathsf{gen}(KB_5)$. To make the notation more compact, we use $Q_4$ for $Q_{hasAdpt,g}$ and $Q_5$ for $Q_{has3DAcc,f}$.

$$\neg Q_4(x) \vee hasAdpt(x, [g(x)]) \tag{7.39}$$
$$\neg Q_5(x) \vee has3DAcc(x, [f(x)]) \tag{7.40}$$

**Saturation of Nonground Closures by $\mathcal{BS}^{\mathbf{D},+}_{DL}$.** We now saturate the nonground closures of $\Xi(KB_5) \cup \mathsf{gen}(KB_5)$ by $\mathcal{BS}^{\mathbf{D},+}_{DL}$. We perform all decomposition inferences, as in this particular example this reduces the total number of inferences, using $Q_6$ for $Q_{hasVD,f}$, and $Q_7$ for $Q_{hasVD,g}$. Conclusions that are tautologies are denoted with †.

$\Rightarrow$ (7.41) $\neg PC(x) \vee \boxed{hasAdpt(x, f(x))}$

$\Rightarrow$ (7.42) $\neg Q_4(x) \vee \boxed{hasAdpt(x, [g(x)])}$

$\Rightarrow$ (7.43) $\neg GrWS(x) \vee \boxed{has3DAcc(x, g(x))}$

$\Rightarrow$ (7.44) $\neg Q_5(x) \vee \boxed{has3DAcc(x, [f(x)])}$

$\Rightarrow$ (7.45) $\boxed{\neg hasAdpt(x, y)} \vee hasVD(x, y)$

---

R(7.41) $\neg Q_6(x) \vee hasVD(x, [f(x)])$
R(7.41) $\neg PC(x) \vee Q_6(x)$
R(7.42) $\neg Q_7(x) \vee hasVD(x, [g(x)])$
R(7.42) $\neg Q_4(x) \vee Q_7(x)$

$\Rightarrow$ (7.46) $\boxed{\neg has3DAcc(x, y)} \vee hasVD(x, y)$

---

R(7.43) $\neg GrWS(x) \vee Q_7(x)$
R(7.44) $\neg Q_5(x) \vee Q_6(x)$

$\Rightarrow$ (7.47) $\boxed{\neg hasAdpt(x, y)} \vee Adpt(y)$

---

R(7.41) $\neg PC(x) \vee Adpt([f(x)])$
R(7.42) $\neg Q_4(x) \vee Adpt([g(x)])$

$\Rightarrow$ (7.48) $\boxed{\neg has3DAcc(x, y)} \vee 3DAcc(y)$

---

R(7.43) $\neg GrWS(x) \vee 3DAcc([g(x)])$
R(7.44) $\neg Q_5(x) \vee 3DAcc([f(x)])$

$\Rightarrow$ (7.49) $\neg Q_6(x) \vee \boxed{hasVD(x, [f(x)])}$

$\Rightarrow$ (7.50) $\neg Q_7(x) \vee \boxed{hasVD(x, [g(x)])}$

$\Rightarrow$ (7.51) $Q_1(x) \vee \boxed{\neg hasVD(x, y)} \vee \neg Adpt3DAcc(y)$

---

R(7.49) $Q_1(x) \vee \neg Q_6(x) \vee \neg Adpt3DAcc([f(x)])$
R(7.50) $Q_1(x) \vee \neg Q_7(x) \vee \neg Adpt3DAcc([g(x)])$

$\Rightarrow$ (7.52) $\neg GaPC(x) \vee \boxed{\neg hasVD(x, y_1)} \vee \boxed{\neg hasVD(x, y_2)} \vee y_1 \approx y_2 \vee \neg VD(y_1) \vee \neg VD(y_2)$

---

R(7.49;7.50) $\neg GaPC(x) \vee \neg Q_6(x) \vee \neg Q_7(x) \vee [g(x)] \approx [f(x)] \vee \neg VD([g(x)]) \vee \neg VD([f(x)])$

$\Rightarrow$ (7.53) $\neg PC(x) \vee \boxed{Adpt([f(x)])}$

$\Rightarrow$ (7.54) $\neg Q_4(x) \vee \boxed{Adpt([g(x)])}$

$\Rightarrow$ (7.55) $\neg GrWS(x) \vee \boxed{3DAcc([g(x)])}$

$\Rightarrow$ (7.56) $\neg Q_5(x) \vee \boxed{3DAcc([f(x)])}$

$\Rightarrow$ (7.57) $\neg Adpt(x) \vee \boxed{VD(x)}$

$\Rightarrow$ (7.58) $\neg 3DAcc(x) \vee \boxed{VD(x)}$

$\Rightarrow$ (7.59) $\neg GaPC(x) \vee \neg Q_6(x) \vee \neg Q_7(x) \vee [g(x)] \approx [f(x)] \vee \boxed{\neg VD([g(x)])} \vee \neg VD([f(x)])$

R(7.57) $\neg GaPC(x) \vee \neg Q_6(x) \vee \neg Q_7(x) \vee [g(x)] \approx [f(x)] \vee \neg Adpt([g(x)]) \vee \neg VD([f(x)])$
R(7.58) $\neg GaPC(x) \vee \neg Q_6(x) \vee \neg Q_7(x) \vee [g(x)] \approx [f(x)] \vee \neg 3DAcc([g(x)]) \vee \neg VD([f(x)])$

$\Rightarrow$ (7.60) $\neg GaPC(x) \vee \neg Q_6(x) \vee \neg Q_7(x) \vee [g(x)] \approx [f(x)] \vee \boxed{\neg Adpt([g(x)])} \vee \neg VD([f(x)])$

R(7.54) $\neg GaPC(x) \vee \neg Q_6(x) \vee \neg Q_7(x) \vee [g(x)] \approx [f(x)] \vee \neg Q_4(x) \vee \neg VD([f(x)])$

$\Rightarrow$ (7.61) $\neg GaPC(x) \vee \neg Q_6(x) \vee \neg Q_7(x) \vee [g(x)] \approx [f(x)] \vee \boxed{\neg 3DAcc([g(x)])} \vee \neg VD([f(x)])$

R(7.55) $\neg GaPC(x) \vee \neg Q_6(x) \vee \neg Q_7(x) \vee [g(x)] \approx [f(x)] \vee \neg GrWS(x) \vee \neg VD([f(x)])$

$\Rightarrow$ (7.62) $\neg GaPC(x) \vee \neg Q_6(x) \vee \neg Q_7(x) \vee \boxed{[g(x)]} \approx [f(x)] \vee \neg Q_4(x) \vee \neg VD([f(x)])$

S(7.43) $\neg GaPC(x) \vee \neg Q_6(x) \vee \neg Q_7(x) \vee Q_5(x) \vee \neg Q_4(x) \vee \neg VD([f(x)])$

$\Rightarrow$ (7.63) $\neg GaPC(x) \vee \neg Q_6(x) \vee \neg Q_7(x) \vee \boxed{[g(x)]} \approx [f(x)] \vee \neg GrWS(x) \vee \neg VD([f(x)])$

S(7.43) $\neg GaPC(x) \vee \neg Q_6(x) \vee \neg Q_7(x) \vee Q_5(x) \vee \neg GrWS(x) \vee \neg VD([f(x)])$

We have performed all inferences involving closures of $KB_1$ containing binary literals; next we perform inferences with unary literals containing a function symbol.

$\Rightarrow$ (7.64) $\neg GaPC(x) \vee \neg Q_6(x) \vee \neg Q_7(x) \vee Q_5(x) \vee \neg Q_4(x) \vee \boxed{\neg VD([f(x)])}$

R(7.57) $\neg GaPC(x) \vee \neg Q_6(x) \vee \neg Q_7(x) \vee Q_5(x) \vee \neg Q_4(x) \vee \neg Adpt([f(x)])$
R(7.58) $\neg GaPC(x) \vee \neg Q_6(x) \vee \neg Q_7(x) \vee Q_5(x) \vee \neg Q_4(x) \vee \neg 3DAcc([f(x)])$

$\Rightarrow$ (7.65) $\neg GaPC(x) \vee \neg Q_6(x) \vee \neg Q_7(x) \vee Q_5(x) \vee \neg GrWS(x) \vee \boxed{\neg VD([f(x)])}$

R(7.57) $\neg GaPC(x) \vee \neg Q_6(x) \vee \neg Q_7(x) \vee Q_5(x) \vee \neg GrWS(x) \vee \neg Adpt([f(x)])$
R(7.58) $\neg GaPC(x) \vee \neg Q_6(x) \vee \neg Q_7(x) \vee Q_5(x) \vee \neg GrWS(x) \vee \neg 3DAcc([f(x)])$

$\Rightarrow$ (7.66) $\neg GaPC(x) \vee \neg Q_6(x) \vee \neg Q_7(x) \vee Q_5(x) \vee \neg Q_4(x) \vee \boxed{\neg Adpt([f(x)])}$

R(7.53) $\neg GaPC(x) \vee \neg Q_6(x) \vee \neg Q_7(x) \vee Q_5(x) \vee \neg Q_4(x) \vee \neg PC(x)$

$\Rightarrow$ (7.67) $\neg GaPC(x) \vee \neg Q_6(x) \vee \neg Q_7(x) \vee Q_5(x) \vee \neg Q_4(x) \vee \boxed{\neg 3DAcc([f(x)])}$

R(7.56)† $\neg GaPC(x) \vee \neg Q_6(x) \vee \neg Q_7(x) \vee Q_5(x) \vee \neg Q_4(x) \vee \neg Q_5(x)$

$\Rightarrow$ (7.68) $\neg GaPC(x) \vee \neg Q_6(x) \vee \neg Q_7(x) \vee Q_5(x) \vee \neg GrWS(x) \vee \boxed{\neg Adpt([f(x)])}$

R(7.53) $\neg GaPC(x) \vee \neg Q_6(x) \vee \neg Q_7(x) \vee Q_5(x) \vee \neg GrWS(x) \vee \neg PC(x)$

$\Rightarrow$ (7.69) $\neg GaPC(x) \vee \neg Q_6(x) \vee \neg Q_7(x) \vee Q_5(x) \vee \neg GrWS(x) \vee \boxed{\neg 3DAcc([f(x)])}$

R(7.56)† $\neg GaPC(x) \vee \neg Q_6(x) \vee \neg Q_7(x) \vee Q_5(x) \vee \neg GrWS(x) \vee \neg Q_5(x)$

$\Rightarrow$ (7.70) $Q_1(x) \vee \neg Q_6(x) \vee \boxed{\neg Adpt3DAcc([f(x)])}$

$\Rightarrow$ (7.71) $Q_1(x) \vee \neg Q_7(x) \vee \boxed{\neg Adpt3DAcc([g(x)])}$

$\Rightarrow$ (7.72) $\neg Adpt(x) \vee \neg 3DAcc(x) \vee Adpt3DAcc(x)$

R(7.70) $\neg Adpt([f(x)]) \vee \neg 3DAcc([f(x)]) \vee Q_1(x) \vee \neg Q_6(x)$
R(7.71) $\neg Adpt([g(x)]) \vee \neg 3DAcc([g(x)]) \vee Q_1(x) \vee \neg Q_7(x)$

$\Rightarrow$ (7.73) $\boxed{\neg Adpt([f(x)])} \vee \neg 3DAcc([f(x)]) \vee Q_1(x) \vee \neg Q_6(x)$

R(7.53) $\neg PC(x) \vee \neg 3DAcc([f(x)]) \vee Q_1(x) \vee \neg Q_6(x)$

$\Rightarrow$ (7.74) $\boxed{\neg Adpt([g(x)])} \vee \neg 3DAcc([g(x)]) \vee Q_1(x) \vee \neg Q_7(x)$

R(7.54) $\neg Q_4(x) \vee \neg 3DAcc([g(x)]) \vee Q_1(x) \vee \neg Q_7(x)$

$\Rightarrow$ (7.75) $\neg PC(x) \vee \boxed{\neg 3DAcc([f(x)])} \vee Q_1(x) \vee \neg Q_6(x)$

R(7.56) $\neg PC(x) \vee \neg Q_5(x) \vee Q_1(x) \vee \neg Q_6(x)$

$\Rightarrow$ (7.76) $\neg Q_4(x) \vee \boxed{\neg 3DAcc([g(x)])} \vee Q_1(x) \vee \neg Q_7(x)$

R(7.55) $\neg Q_4(x) \vee \neg GrWS(x) \vee Q_1(x) \vee \neg Q_7(x)$

Next we perform the inferences with closures from $KB_3$ containing function symbols:

$\Rightarrow$ (7.77) $\neg Q_2(x) \vee \boxed{price(x, h(x))}$

$\Rightarrow$ (7.78) $\neg Q_3(x) \vee \boxed{price(x, i(x))}$

$\Rightarrow$ (7.79) $\boxed{\neg price(x, y_1)} \vee \boxed{\neg price(x, y_2)} \vee y_1 \approx_{\mathbf{D}} y_2$

R(7.77;7.78) $\neg Q_2(x) \vee \neg Q_3(x) \vee [h(x)] \approx_{\mathbf{D}} [i(x)]$

$\Rightarrow$ (7.80) $\neg Q_2(x) \vee \boxed{\geq_{2000}(h(x))}$

$\Rightarrow$ (7.81) $\neg Q_3(x) \vee \boxed{\leq_{1500}(i(x))}$

$\Rightarrow$ (7.82) $\neg Q_2(x) \vee \neg Q_3(x) \vee \boxed{[h(x)] \approx_{\mathbf{D}} [i(x)]}$

D(7.80;7.81) $\neg Q_2(x) \vee \neg Q_3(x)$

We recapitulate in the following table the closures from the unused set at this point. Obviously, all closures produced by remaining inferences do not contain function symbols; such closures are irrelevant in the sense of Definition 7.3.1, and, by Lemma 7.3.2, are not needed in $\mathsf{DD}(KB_5)$. Therefore, we do not show the remaining inferences.

$$\neg GrWS(x) \vee PC(x) \tag{7.83}$$
$$\neg GaPC(x) \vee PC(x) \tag{7.84}$$
$$\neg GaPC(x) \vee GrWS(x) \tag{7.85}$$
$$\neg PC(x) \vee Q_6(x) \tag{7.86}$$
$$\neg Q_4(x) \vee Q_7(x) \tag{7.87}$$
$$\neg GrWS(x) \vee Q_7(x) \tag{7.88}$$
$$\neg Q_5(x) \vee Q_6(x) \tag{7.89}$$
$$\neg GaPC(x) \vee \neg Q_6(x) \vee \neg Q_7(x) \vee Q_5(x) \vee \neg Q_4(x) \vee \neg PC(x) \tag{7.90}$$
$$\neg GaPC(x) \vee \neg Q_6(x) \vee \neg Q_7(x) \vee Q_5(x) \vee \neg GrWS(x) \vee \neg PC(x) \tag{7.91}$$
$$\neg PC(x) \vee \neg Q_5(x) \vee Q_1(x) \vee \neg Q_6(x) \tag{7.92}$$
$$\neg Q_4(x) \vee \neg GrWS(x) \vee Q_1(x) \vee \neg Q_7(x) \tag{7.93}$$
$$\neg HighEnd(x) \vee GrWS(x) \vee Q_2(x) \tag{7.94}$$
$$\neg Q_2(x) \vee PC(x) \tag{7.95}$$
$$\neg Q_2(x) \vee \neg Q_3(x) \tag{7.96}$$

**Conversion to Disjunctive Datalog.** The saturated set of closures is converted into a disjunctive datalog program by eliminating function symbols (see Section 7.2), eliminating irrelevant clauses (see Section 7.3), and translating the obtained clauses to the rule form (see Section 7.4).

To understand how to eliminate function symbols, consider the closure (7.41). It contains the term $f(x)$. This term is replaced with a new variable $x_f$ and a literal $\neg S_f(x, x_f)$ is appended. Then, the positive literals are moved to the rule head, and the negative literals to the rule body, yielding the rule (7.97). The following rules are obtained by applying the process:

$$hasAdpt(x, x_f) \leftarrow PC(x), S_f(x, x_f) \tag{7.41}$$
$$hasAdpt(x, x_g) \leftarrow Q_4(x), S_g(x, x_g) \tag{7.42}$$
$$has3DAcc(x, x_g) \leftarrow GrWS(x), S_g(x, x_g) \tag{7.43}$$
$$has3DAcc(x, x_f) \leftarrow Q_5(x), S_f(x, x_f) \tag{7.44}$$
$$hasVD(x, y) \leftarrow hasAdpt(x, y) \tag{7.45}$$
$$hasVD(x, y) \leftarrow has3DAcc(x, y) \tag{7.46}$$
$$Adpt(y) \leftarrow hasAdpt(x, y) \tag{7.47}$$
$$3DAcc(y) \leftarrow has3DAcc(x, y) \tag{7.49}$$
$$hasVD(x, x_f) \leftarrow Q_6(x), S_f(x, x_f) \tag{7.49}$$
$$hasVD(x, x_g) \leftarrow Q_7(x), S_g(x, x_g) \tag{7.50}$$
$$Q_1(x) \leftarrow hasVD(x, y), Adpt3DAcc(y) \tag{7.51}$$
$$y_1 \approx y_2 \leftarrow GaPC(x), hasVD(x, y_1), hasVD(x, y_2), VD(y_1), VD(y_2) \tag{7.52}$$

Consider the closure (7.53); to understand why this closure is irrelevant, we next show the $\mathcal{BS}^{\mathbf{D},+}_{DL}$ inference deriving the closure.

$$\frac{\neg PC(x) \vee \boxed{hasAdpt(x, f(x))} \quad \boxed{\neg hasAdpt(x_1, y_1)} \vee Adpt(y_1)}{\neg PC(x) \vee Adpt([f(x)])}$$

The most general unifier used in the inference is $\sigma = \{x_1 \mapsto x, y_1 \mapsto f(x)\}$. The result of applying $\sigma$ and $\lambda$ to the clauses is as follows:

$$\frac{\neg PC(x) \vee \boxed{hasAdpt(x, f(x))} \quad \boxed{\neg hasAdpt(x, [f(x)])} \vee Adpt([f(x)])}{\neg PC(x) \vee Adpt([f(x)])}$$

$$\Downarrow_\lambda$$

$$\frac{\neg S_f(x, x_f) \vee \neg PC(x) \vee \boxed{hasAdpt(x, x_f)} \quad \boxed{\neg hasAdpt(x, x_f)} \vee Adpt(x_f) \vee \neg S_f(x, x_f)}{\neg PC(x) \vee Adpt(x_f) \vee \neg S_f(x, x_f)}$$

Now all variables in the inference premises occur in the inference conclusion. Also, all premises are members of the set of saturated closures. It is now clear that (7.53) is irrelevant, since it fulfills the conditions of Definition 7.3.1. In a similar way one can see that all closures (7.54)–(7.56) are irrelevant. Intuitively, a closure $C$ derived by an inference $\xi$ from premises $P_i$ is irrelevant if $\lambda(C)$ is obtained from $\lambda(P_i)$ by the inference $\lambda(\xi)$; in such a case, the premises $\lambda(P_i)$ imply the conclusion $\lambda(C)$. In the previous example this is obviously the case, so by removing (7.53) we perform "unresolution."

We contrast this with the relevant closure (7.93). The inference deriving the closure is as follows:

$$\frac{\neg Q_4(x) \vee \boxed{\neg 3DAcc([g(x)])} \vee Q_1(x) \vee \neg Q_7(x) \quad \neg GrWS(x_1) \vee \boxed{3DAcc([g(x_1)])}}{\neg Q_4(x) \vee \neg GrWS(x) \vee Q_1(x) \vee \neg Q_7(x)}$$

The most general unifier is $\sigma = \{x_1 \mapsto x\}$; we now show the effects of applying $\sigma$ and $\lambda$ to the inference:

$$\frac{\neg Q_4(x) \vee \boxed{\neg 3DAcc([g(x)])} \vee Q_1(x) \vee \neg Q_7(x) \quad \neg GrWS(x) \vee \boxed{3DAcc([g(x)])}}{\neg Q_4(x) \vee \neg GrWS(x) \vee Q_1(x) \vee \neg Q_7(x)}$$

$$\Downarrow_\lambda$$

$$\frac{\neg S_g(x, x_g) \vee \neg Q_4(x) \vee \boxed{\neg 3DAcc(x_g)} \vee Q_1(x) \vee \neg Q_7(x) \quad \neg GrWS(x) \vee \boxed{3DAcc(x_g)} \vee \neg S_g(x, x_g)}{\neg Q_4(x) \vee \neg GrWS(x) \vee Q_1(x) \vee \neg Q_7(x)}$$

In this case, $\lambda(C)$ is not obtained by the resolution of $\lambda(P_1)$ and $\lambda(P_2)$, since $\lambda(C)$ does not contain $\neg S_g(x, x_g)$. The inference deriving (7.93) eliminates the function symbol $g$ from the premises. Such inferences are crucial in the reduction, since they ensure that the obtained disjunctive datalog program entails the same set of ground facts as the original knowledge base.

Note that, although irrelevant closures are removed in the translation to disjunctive datalog, they still have to be derived during saturation. Namely, the irrelevant closure (7.53) is used (indirectly) in the saturation to derive the relevant closure (7.93).

We now translate the remaining (relevant) closures into disjunctive datalog rules.

(7.109) $VD(x) \leftarrow Adpt(x)$ (7.57)
(7.110) $VD(x) \leftarrow 3DAcc(x)$ (7.58)
(7.111) $Q_5(x) \leftarrow GaPC(x), Q_6(x), Q_7(x), Q_4(x), VD(x_f), S_f(x, x_f)$ (7.64)
(7.112) $Q_5(x) \leftarrow GaPC(x), Q_6(x), Q_7(x), GrWS(x), VD(x_f), S_f(x, x_f)$ (7.65)
(7.113) $Adpt3DAcc(x) \leftarrow Adpt(x), 3DAcc(x)$ (7.72)
(7.114) $price(x, x_h) \leftarrow Q_2(x), S_h(x, x_h)$ (7.77)
(7.115) $price(x, x_i) \leftarrow Q_3(x), S_i(x, x_i)$ (7.78)
(7.116) $y_1 \approx_{\mathbf{D}} y_2 \leftarrow price(x, y_1), price(x, y_2)$ (7.79)
(7.117) $\geq_{2000}(x_h) \leftarrow Q_2(x), S_h(x, x_h)$ (7.80)
(7.118) $\leq_{1500}(x_i) \leftarrow Q_3(x), S_i(x, x_i)$ (7.81)
(7.119) $PC(x) \leftarrow GrWS(x)$ (7.83)
(7.120) $PC(x) \leftarrow GaPC(x)$ (7.84)
(7.121) $GrWS(x) \leftarrow GaPC(x)$ (7.85)
(7.122) $Q_6(x) \leftarrow PC(x)$ (7.86)
(7.123) $Q_7(x) \leftarrow Q_4(x)$ (7.87)
(7.124) $Q_7(x) \leftarrow GrWS(x)$ (7.88)
(7.125) $Q_6(x) \leftarrow Q_5(x)$ (7.89)
(7.126) $Q_5(x) \leftarrow GaPC(x), Q_6(x), Q_7(x), Q_4(x), PC(x)$ (7.90)
(7.127) $Q_5(x) \leftarrow GaPC(x), Q_6(x), Q_7(x), GrWS(x), PC(x)$ (7.91)
(7.128) $Q_1(x) \leftarrow PC(x), Q_5(x), Q_6(x)$ (7.92)
(7.129) $Q_1(x) \leftarrow Q_4(x), GrWS(x), Q_7(x)$ (7.93)
(7.130) $GrWS(x) \vee Q_2(x) \leftarrow HighEnd(x)$ (7.94)
(7.131) $PC(x) \leftarrow Q_2(x)$ (7.95)
(7.132) $\leftarrow Q_2(x), Q_3(x)$ (7.96)

We finally append the ABox clauses:

(7.133) $GaPC(pc_1)$ (7.28)
(7.134) $Q_3(pc_2)$ (7.37)
(7.135) $HighEnd(pc_2)$ (7.38)
(7.136) $S_f(pc_1, pc_{1\,f})$
(7.137) $S_f(pc_2, pc_{2\,f})$
(7.138) $S_g(pc_1, pc_{1\,g})$
(7.139) $S_g(pc_2, pc_{2\,g})$
(7.140) $S_h(pc_1, pc_{1\,h})$
(7.141) $S_h(pc_2, pc_{2\,h})$
(7.142) $S_i(pc_1, pc_{1\,i})$
(7.143) $S_i(pc_2, pc_{2\,i})$

We now make $\mathsf{DD}(KB_5)$ to be a program consisting of rules (7.97)–(7.143). In this program, of all predicates $Q_*$, the predicate $Q_1$ is introduced for query answering, and the predicate $Q_3$ occurs in the ABox. The predicates $Q_2$, $Q_4$, $Q_5$, $Q_6$ and $Q_7$ are introduced by decomposition, and are not of interest in query answering. Therefore, to reduce the size of the program, it is possible to apply the standard rule-unfolding strategy. Here we refrain from doing so for the sake of brevity.

**Answering Queries in $\mathsf{DD}(KB_5)$.** The program $\mathsf{DD}(KB_5)$ can be used to answer queries. We show that $KB_5 \models \exists hasVD.Adpt3DAcc(pc_1)$, see Subsection 3.1.1, by demonstrating that $\mathsf{DD}(KB_5) \models_c Q_1(pc_1)$. Since $\mathsf{DD}(KB_5)$ is large, we do not show all inferences in the bottom-up saturation, but only those leading to the goal.

| | | |
|---|---|---|
| (7.144) | $PC(pc_1)$ | R(7.120;7.133) |
| (7.145) | $GrWS(pc_1)$ | R(7.121;7.133) |
| (7.146) | $Q_6(pc_1)$ | R(7.122;7.144) |
| (7.147) | $Q_7(pc_1)$ | R(7.124;7.145) |
| (7.148) | $Q_5(pc_1)$ | R(7.127;7.133;7.146;7.147;7.145;7.144) |
| (7.149) | $Q_1(pc_1)$ | R(7.128;7.144;7.148;7.146) |

In a similar way, we show that $KB_5 \models GrWS(pc_2)$, see Subsection 3.2.3. The query $GrWS(x)$ cannot be answered directly, since the predicate $GrWS$ occurs in the body of several rules, so we add a new rule (7.150). We use an ordering as in Section 7.6, with $Q_8$ being the smallest predicate.

| | | |
|---|---|---|
| (7.150) | $Q_8(x) \leftarrow GrWS(x)$ | |
| (7.151) | $\boxed{GrWS(pc_2)} \vee Q_2(pc_2)$ | R(7.130;7.135) |
| (7.152) | $Q_8(pc_2) \vee \boxed{Q_2(pc_2)}$ | R(7.150;7.151) |
| (7.153) | $Q_8(pc_2)$ | R(7.132;7.152;7.134) |

## 7.8 Related Work

Our work was largely motivated by [57], where the authors investigated a decidable intersection of description logic and logic programming. In particular, the authors identify the description logic constructs that can be encoded and executed using existing rule engines. Thus, the description logic component allows only existential quantifiers to occur under negative, and universal quantifiers to occur under positive polarity. The authors present an operator for translating a description logic knowledge base into a logic program. This approach was later extended in [148] to support more expressive description logic, provided that the rule engine supports advanced features. However, our approach is a significant extension since we handle full $\mathcal{SHIQ}(\mathbf{D})$.

In [64], it was shown how to convert $\mathcal{SHIQ}^*$ knowledge bases into *conceptual logic programs* (CLP). CLPs generalize the good properties of description logic to the framework of answer set programming. Apart from the usual constructs, $\mathcal{SHIQ}^*$ supports the transitive closure of roles. Although the presented transformation preserves the semantics of the knowledge base, the obtained answer set program is not safe. Hence, its grounding is infinite, so the program cannot be evaluated using existing answer set solvers. The problem of decidable reasoning for CLPs is addressed by an automata-based technique. On the contrary, our transformation produces a safe program with

a finite grounding. Hence, issues related to decidability of reasoning are handled by the transformation, and not by the query answering algorithm: the disjunctive program obtained by our approach can be evaluated using any technique for reasoning in disjunctive datalog programs.

Another approach for reducing description logic knowledge bases to answer set programming was presented in [2]. To deal with existential quantification, this approach uses function symbols. Thus, the Herbrand universe of the programs obtained by the reduction is infinite, so existing answer set solvers cannot be used for reasoning. In fact, decidability is not considered at all.

The query answering algorithm from Section 7.6 was inspired by [25], where the authors suggested that the fixpoint computation of disjunctive semantics can be optimized by introducing an appropriate literal ordering. Our approach extends the one from [25] by showing that redundancy elimination techniques can be applied in the fixpoint computation. Both algorithms are based on the idea of using answer literals to compute answers, which was first proposed in [56].

As discussed in Section 7.6, the state-of-the-art technique for reasoning in disjunctive datalog is intelligent grounding [43]. It is actually a nondeterministic model building technique. Ground query answering problems are reduced to model building, and nonground query answering is solved by grounding the query and checking each ground answer separately. Our query answering algorithm from Section 7.6 is not based on model building. It does not require grounding the disjunctive program, it computes all answers to nonground queries in one pass, and does not require nondeterministic guessing. However, it requires exponential space in the worst case, whereas intelligent ground requires only polynomial space.

# Chapter 8

# Data Complexity of Reasoning

Algorithms from Chapters 5, 6, and 7 run in worst-case exponential time in $|KB|$, assuming unary coding of numbers, a bound on the arity of concrete predicates, and an exponential oracle for reasoning with a concrete domain. In [144], $\mathcal{SHIQ}$ was shown to be ExpTime-complete for any coding of numbers, so our algorithms are (almost) worst-case optimal. However, our complexity results, as well as the results from [144], actually address the *combined complexity* of reasoning, which is measured in the size of the entire knowledge base $KB$, including the RBox, the TBox, and the ABox.

ExpTime-completeness is a rather discouraging result, since $|KB|$ can be large in practice. However, by drawing a parallel with traditional database applications, $|KB|$ often depends mainly on the size of the ABox, while the sizes of the TBox and the RBox are usually negligible. Namely, in many applications of description logics, the TBox and the RBox are used as the database schema, whereas the ABox is used as the database instance. For such applications, a much better performance estimate is given by *data complexity*—the complexity measured only in $|KB_{\mathcal{A}}|$, under the assumption that $|KB_{\mathcal{R}} \cup KB_{\mathcal{T}}|$ is bound by a constant. In practice, data complexity provides a good performance estimate if $|KB_{\mathcal{R}}| + |KB_{\mathcal{T}}| = |KB_{\mathcal{TR}}| \ll |KB_{\mathcal{A}}|$.

To fully separate the terminological from the assertional knowledge, we assume in this chapter that $KB$ is extensionally reduced; that is, the ABox axioms of $KB$ contain only (negations of) atomic concepts. Namely, complex concept axioms in ABox assertions actually specify terminological knowledge, so, for extensionally reduced knowledge bases, $|KB_{\mathcal{A}}|$ is the measure of raw input data processed by the algorithm.

To the best of our knowledge, data complexity of testing knowledge base satisfiability even for the basic logic $\mathcal{ALC}$ with general TBoxes was previously unknown. Based on the results from Chapter 7, in Section 8.1 we show that satisfiability checking is NP-complete in the size of ABox for any logic between $\mathcal{ALC}$ and $\mathcal{SHIQ}(\mathbf{D})$ (regardless of how numbers are encoded), and that instance and role checking—the basic query answering problems for description logics—are co-NP-complete.

To obtain a formalism with an even better data complexity, in Section 8.2 we define Horn-$\mathcal{SHIQ}(\mathbf{D})$—a logic related to $\mathcal{SHIQ}(\mathbf{D})$ analogously as Horn logic is related to

full first-order logic. Namely, Horn-$\mathcal{SHIQ}(\mathbf{D})$ provides existential and universal quantifiers, but does not provide for modeling disjunctive knowledge. This restriction allows us to show that basic reasoning problems for Horn-$\mathcal{SHIQ}(\mathbf{D})$ are P-complete in $|KB_{\mathcal{A}}|$. Hence, along the traditional lines of knowledge representation research, the capability of representing disjunctive information is traded for polynomial data complexity. Since disjunctive reasoning is not needed for many applications, Horn-$\mathcal{SHIQ}(\mathbf{D})$ is a very appealing logic because it is still very expressive, but there is theoretical evidence that it can be implemented efficiently in practice. Furthermore, Horn-$\mathcal{SHIQ}(\mathbf{D})$ extends DL-lite [27], a logic aiming to capture most constructs of formalisms for conceptual modeling, such as Entity–Relationship (ER) diagrams or the Unified Modeling Language (UML), while providing polynomial algorithms for satisfiability checking and conjunctive query answering (assuming an unbounded knowledge base, but a bound on the query size). Horn-$\mathcal{SHIQ}(\mathbf{D})$ additionally provides for concrete predicates, qualified existential quantification, conditional functionality and role inclusions, while providing a reasoning algorithm polynomial in the size of data.

To develop an intuition and to provide a more detailed account behind our results, in Section 8.3 we compare our results with the similar results for datalog and its variants [35].

## 8.1 Data Complexity of Satisfiability

Our results from Chapter 7 show almost immediately that checking satisfiability of a $\mathcal{SHIQ}(\mathbf{D})$ knowledge base $KB$ is NP-complete in the size of data.

**Lemma 8.1.1** (Membership)**.** *For an extensionally reduced $\mathcal{SHIQ}(\mathbf{D})$ knowledge base $KB$, data complexity of checking $\mathbf{D}$-satisfiability of $KB$ is in* NP*, assuming a bound on the arity of concrete predicates and that $\mathbf{D}$-satisfiability of finite conjunctions over $\Phi_{\mathbf{D}}$ can be decided in polynomial time.*

*Proof.* Let $c$ be the number of constants, $f$ the number of function symbols in the signature of $\Xi(KB)$, and $s$ the number of facts in $\Xi(KB)$. By Definition 7.2.2, the number of constants in $\mathsf{DD}(KB)$ is bounded by $\ell_1 = c + cf$ ($cf$ accounts for constants of the form $a_f$), and the number of facts in $\mathsf{DD}(KB)$ is bounded by $\ell_2 = s + c + 2cf$ ($c$ accounts for facts of the form $HU(a)$, one $cf$ accounts for the facts of the form $S_f(a, a_f)$, and the other $cf$ accounts for the facts of the form $HU(a_f)$). Since we assume that $|KB_{\mathcal{TR}}|$ is bounded by a constant, $f$ is bounded by a constant (regardless of how numbers are encoded), so both $\ell_1$ and $\ell_2$ are linear in $|KB_{\mathcal{A}}|$.

Hence, $|\mathsf{DD}(KB)|$ is exponential in $|KB|$ only because the number of rules in $\mathsf{DD}(KB)$ is bounded by the number of closures obtained by saturating the set of closures $\Gamma_{\mathcal{TR}g} = \Xi(KB_{\mathcal{T}} \cup KB_{\mathcal{R}}) \cup \mathsf{gen}(KB)$ by $\mathcal{BS}_{DL}^{\mathbf{D},+}$. Since $\Gamma_{\mathcal{TR}g}$ does not contain ABox closures, by Lemma 5.3.9, the number of closures in $\mathsf{Sat}(\Gamma_{\mathcal{TR}g})$ is exponential in $|KB_{\mathcal{TR}}|$. Since we assume that the latter is bounded by a constant, both the number of rules in $\mathsf{DD}(KB)$ and their length are bounded by constants, so $|\mathsf{DD}(KB)|$ is linear in $|KB_{\mathcal{A}}|$, and can be computed from $KB$ in linear time. Now the claim of the lemma

follows from a well-known fact that checking satisfiability of a disjunctive datalog program is NP-complete [35]. A minor difference is that $\mathsf{DD}(KB)$ can contain concrete predicates, so for the sake of completeness we prove this result as well.

The program $\mathsf{DD}(KB)$ can be $\mathsf{c}$-factored in time linear in $|KB_{\mathcal{A}}|$, so it suffices to show that the result is $\mathbf{d}$-satisfiable. Let $n$ be the number of rules introduced by $\mathsf{c}$-factoring; obviously, $n$ is linear in $|KB_{\mathcal{A}}|$. Assuming that $\mathsf{DD}(KB)$ contains $r$ rules with at most $v$ variables per rule, the number of literals in a ground instantiation $\mathsf{ground}(\mathsf{DD}(KB))$ is bounded by $r \cdot \ell_1^v + n \cdot \ell_1 + \ell_2$ (in each rule, each variable can be replaced in $\ell_1$ possible ways). Assuming $r$ and $v$ are bounded by constants, $|\mathsf{ground}(\mathsf{DD}(KB))|$ is polynomial in $|KB_{\mathcal{A}}|$. Now $\mathbf{d}$-satisfiability of $\mathsf{ground}(\mathsf{DD}(KB))$ can be checked by nondeterministically generating an interpretation $I$, and then checking whether it is a $\mathbf{d}$-model. Checking whether $I$ is a model can be performed in polynomial time. To additionally check whether $I$ is a $\mathbf{d}$-model, it is sufficient to check whether $\widehat{S}$ is $\mathbf{D}$-satisfiable, where $S$ is the subset of $I$ containing all literals with a concrete predicate. This can be done in polynomial time by assumption. Since $\mathsf{DD}(KB)$ and $KB$ are $\mathbf{D}$-equisatisfiabile by Theorem 7.4.2, the claim of the lemma follows. □

Schaerf showed in [126, Lemma 4.2.7] that the problem of instance checking in a very simple logic $\mathcal{ALE}$ is co-NP-hard in the size of data. The proof employs a reduction from 2-2-SATISFIABILITY—the problem of checking satisfiability of CNF formulae where each clause contains exactly two positive and two negative literals—, which is known to be NP-complete. This reduction produces an extensionally reduced knowledge base, so it immediately provides a lower bound for data complexity of satisfiability checking. For the sake of completeness, we give a simple, alternative hardness proof.

**Lemma 8.1.2** (Hardness)**.** *Checking satisfiability of an $\mathcal{ALC}$ knowledge base $KB$ is* NP*-hard in $|KB_{\mathcal{A}}|$.*

*Proof.* The proof is by the reduction from 3-GRAPH COLORING: a graph $G$ is said to be 3-*colorable* if and only if it is possible to assign a singe color from the set $\{Red, Green, Blue\}$ to each of the graph's vertices such that no two adjoining vertices have the same color. Deciding whether $G$ is 3-colorable is NP-complete [104].

For a graph $G$, we construct the knowledge base $KB_G$, whose ABox contains the assertions $edge(e, f)$ and $edge(f, e)$ for each edge $\langle e, f\rangle$ in $G$, and whose TBox contains the following axioms:

$$\top \sqsubseteq Red \sqcup Green \sqcup Blue \tag{8.1}$$

$$Red \sqcap Green \sqsubseteq \bot \tag{8.2}$$

$$Green \sqcap Blue \sqsubseteq \bot \tag{8.3}$$

$$Red \sqcap Blue \sqsubseteq \bot \tag{8.4}$$

$$Red \sqsubseteq \forall edge.\neg Red \tag{8.5}$$

$$Green \sqsubseteq \forall edge.\neg Green \tag{8.6}$$

$$Blue \sqsubseteq \forall edge.\neg Blue \tag{8.7}$$

Obviously, $G$ is 3-colorable if and only if $KB_G$ is satisfiable. Namely, (8.1) states that each individual (that is, each vertex) in a model is assigned at least one color; (8.2)–(8.4) specify that colors are pair-wise disjoint so each vertex is assigned at most one color; and (8.5)–(8.7) specify the conditions of 3-colorability. Hence, each model of $KB_G$ gives an assignment of colors to vertices of $G$; conversely, each assignment of colors of $G$ defines a model of $KB_G$. Since the size of the TBox of $KB_G$ is constant, and the size of ABox of $KB_G$ is double the number of edges of $G$, the claim of the lemma follows. □

We now state the main result of this section.

**Theorem 8.1.3.** *Let $KB$ be an extensionally reduced knowledge base expressed in any logic between $\mathcal{ALC}$ and $\mathcal{SHIQ}(\mathbf{D})$. Assuming a polynomial oracle for reasoning with concrete domains and a bound on the arity of concrete domain predicates, the following claims hold:*

- *Checking $KB$ (**D**-)satisfiability is data complete for* NP.
- *Checking $KB$ (**D**-)unsatisfiability is data complete for co-*NP.
- *Checking whether $KB \models_{(\mathbf{D})} \alpha$, for $\alpha$ of the form $(\neg)C(a)$ with $|C|$ bounded or of the form $(\neg)R(a, b)$, is also data complete for co-*NP.

*Proof.* The first claim is a direct consequence of lemmata 8.1.1 and 8.1.2. Checking (**D**-)unsatisfiability is a complementary problem to checking (**D**-)satisfiability, so the second claim follows from the first one. Furthermore, both instance checking (assuming $|C|$ is bounded) and role checking can be reduced to (**D**-)unsatisfiability in constant time, so the third claim follows from the second. □

We finish this section with a remark that, in an extensionally reduced knowledge base, ABox assertions do not contain number restrictions, so $|KB_{\mathcal{A}}|$ does not depend on the used number coding. Hence, the results of Theorem 8.1.3 also do not depend on the coding of numbers.

## 8.2 A Horn Fragment of $\mathcal{SHIQ}(\mathbf{D})$

Horn logic is a well-known fragment of first-order logic in which formulae are restricted to clauses containing at most one positive literal. Because of this restriction, a Horn clause can be understood as a rule in which a conjunction of several body literals imply one head literal. The main limitation of Horn logic is the inability to represent disjunctive information; however, its main benefit are practical refutation procedures, such as SLD-resolution [109]. Furthermore, if function symbols are not used, query answering in Horn logic is data complete for P [145, 78, 35], thus making function-free Horn logic appealing for practical usage.

Following this idea, in this section we identify the Horn fragment of $\mathcal{SHIQ}(\mathbf{D})$ that provides similar properties. In Horn-$\mathcal{SHIQ}(\mathbf{D})$, the capability of representing disjunctive information is traded for P-complete data complexity of reasoning. Roughly speaking, only TBox axioms of the form $\bigsqcap C_i \sqsubseteq D$ are allowed, where $C_i$ is $A$, $\exists R.A$, or $\geq n\, R.A$, and $D$ is $A$, $\bot$, $\exists R.A$, $\forall R.A$, $\geq n\, R.A$, or $\leq 1\, R$; additionally, Horn-$\mathcal{SHIQ}(\mathbf{D})$ allows role inclusion axioms and, in certain situations, transitivity axioms, as discussed after Definition 8.2.1. Whereas such a definition succinctly demonstrates the expressivity of the fragment, it is too restricting in general. For example, the axiom $A_1 \sqcup A_2 \sqsubseteq \neg B$ is not Horn, but it is equivalent to Horn axioms $A_1 \sqcap B \sqsubseteq \bot$ and $A_2 \sqcap B \sqsubseteq \bot$. Similarly, a non-Horn axiom $A \sqsubseteq \exists R.(\exists R.B)$ can be transformed into Horn axioms $A \sqsubseteq \exists R.Q$ and $Q \sqsubseteq \exists R.B$ by introducing a new name $Q$ for the subconcept $\exists R.B$. To avoid dependency on such obvious syntactic transformations, we use the following, rather technical definition of Horn-$\mathcal{SHIQ}(\mathbf{D})$:

**Definition 8.2.1.** *Let* $\mathsf{pl}^+$ *and* $\mathsf{pl}^-$ *be two mutually recursive functions assigning a nonnegative integer to a* $\mathcal{SHIQ}(\mathbf{D})$ *concept* $D$*, defined inductively as in Table 8.1, where* $\mathsf{sgn}(0) = 0$*, and* $\mathsf{sgn}(n) = 1$ *for* $n > 0$*. For a concept* $D$ *and a position* $p$ *of a subconcept in* $D$*, let* $\mathsf{pl}(D, p)$ *be defined as follows:*[1]

$$\mathsf{pl}(D,p) = \begin{cases} \mathsf{pl}^+(D|_p) & \text{if } \mathsf{pol}(D,p) = 1 \\ \mathsf{pl}^-(D|_p) & \text{if } \mathsf{pol}(D,p) = -1 \end{cases}$$

*A* $\mathcal{SHIQ}(\mathbf{D})$ *concept* $C$ *is a* Horn *concept if* $\mathsf{pl}(C,p) \leq 1$ *for each position* $p$ *of a subconcept in* $C$ *(including the empty position* $\epsilon$*). An extensionally reduced* $\mathcal{ALCHIQ}(\mathbf{D})$ *knowledge base* $KB$ *is* Horn *if, for each axiom* $C \sqsubseteq D \in KB_{\mathcal{A}}$*, the concept* $\neg C \sqcup D$ *is Horn. An extensionally reduced* $\mathcal{SHIQ}(\mathbf{D})$ *knowledge base* $KB$ *is* Horn *if* $\Omega(KB)$ *is Horn.*

Observe that, for a concept $C$ without nonatomic subconcepts, $\mathsf{pl}^+(C)$ yields the maximal number of positive literals in clauses obtained by clausifying $\forall x : \pi_y(C, x)$. To clausify a concept $C$ containing a nonatomic subconcept at position $p$, one should consider if $C|_p$ occurs in $C$ under positive or negative polarity. For example, in a concept $\neg(\neg A \sqcap \neg B)$, the concepts $A$ and $B$ actually occur positively, and $\sqcap$ is actually $\sqcup$. Hence, $\mathsf{pl}^+(C|_p)$ ($\mathsf{pl}^-(C|_p)$) counts the number of positive literals used to clausify $C|_p$, provided that $C|_p$ occurs in $C$ under positive (negative) polarity.

The function $\mathsf{sgn}(\cdot)$ takes into account that $C|_p$ is replaced in $C$ by structural transformation with only one atomic concept, even if clausification of $C|_p$ produces more than one positive literal. For example, to clausify $C = \forall R.(D_1 \sqcup D_2)$, the structural transformation replaces $D_1 \sqcup D_2$ with a new atomic concept $Q$, yielding $C' = \forall R.Q$; now clausifying $C'$ produces a clause with only one positive literal.

A concept $C$ is Horn if the maximal number of positive literals obtained by clausifying subconcepts of $C$ is at most one. Recall that, although transitivity axioms are translated by $\pi$ into Horn clauses, the algorithm from Section 5.2 replaces them by

[1] The abbreviation $\mathsf{pl}$ stems from "[the number of] positive literals."

Table 8.1: Definitions of $\mathsf{pl}^+$ and $\mathsf{pl}^-$

| $D$ | $\mathsf{pl}^+(D)$ | $\mathsf{pl}^-(D)$ |
|---|---|---|
| $\top$ | 0 | 0 |
| $\bot$ | 0 | 0 |
| $A$ | 1 | 0 |
| $\neg C$ | $\mathsf{pl}^-(C)$ | $\mathsf{pl}^+(C)$ |
| $C_1 \sqcap \ldots \sqcap C_n$ | $\max_{1\leq i\leq n} \mathsf{sgn}(\mathsf{pl}^+(C_i))$ | $\sum_{1\leq i\leq n} \mathsf{sgn}(\mathsf{pl}^-(C_i))$ |
| $C_1 \sqcup \ldots \sqcup C_n$ | $\sum_{1\leq i\leq n} \mathsf{sgn}(\mathsf{pl}^+(C_i))$ | $\max_{1\leq i\leq n} \mathsf{pl}^-(C_i)$ |
| $\exists R.C$ | 1 | $\mathsf{sgn}(\mathsf{pl}^-(C))$ |
| $\forall R.C$ | $\mathsf{sgn}(\mathsf{pl}^+(C))$ | 1 |
| $\geq n\, R.C$ | 1 | $\frac{(n-1)n}{2} + n \cdot \mathsf{sgn}(\mathsf{pl}^-(C))$ |
| $\leq n\, R.C$ | $\frac{n(n+1)}{2} + (n+1) \cdot \mathsf{sgn}(\mathsf{pl}^-(C))$ | 1 |
| $\exists T_1, \ldots, T_m.d$ | 1 | 1 |
| $\forall T_1, \ldots, T_m.d$ | 1 | 1 |
| $\geq n\, T$ | 1 | $\frac{(n-1)n}{2}$ |
| $\leq n\, T$ | $\frac{n(n+1)}{2}$ | 1 |

axioms of the form $\forall R.C \sqsubseteq \forall S.(\forall S.C)$. Now $\mathsf{pl}^+(\exists R.\neg C \sqcup \forall S.(\forall S.C)) = 1 + \mathsf{pl}^+(C)$, so if $\mathsf{pl}^+(C) > 0$, $\Omega(KB)$ is not a Horn knowledge base. Hence, the presence of transitivity axioms can make $KB$ non-Horn.

If a concept $C$ has a nonatomic subconcept at position $p$, special care has to be taken when introducing a new name $\alpha$ for $C|_p$. For example, consider the Horn concept $C = \forall R.D_1 \sqcup \forall R.\neg D_2$. To apply the structural transformation to $C$, one might replace $\forall R.D_1$ and $\forall R.\neg D_2$ with new atomic concepts $Q_1$ and $Q_2$, yielding $\neg Q_1 \sqcup \forall R.D_1$, $\neg Q_2 \sqcup \forall R.\neg D_2$, and $Q_1 \sqcup Q_2$. The problem is that the concept $C$ is Horn, but $Q_1 \sqcup Q_2$ is not. The previous example shows that a straightforward application of the structural transformation can destroy Hornness. To remedy this, we modify the structural transformation to replace each $C|_p$ with $\alpha$ chosen in a way such that clausifying $\alpha$ and $C|_p$ requires the same number of positive literals. In the previous example, this would mean that $\forall R.D_1$ should be replaced with $Q_1$, but $\forall R.\neg D_2$ should be replaced with $\neg Q_2$, yielding concepts $\neg Q_1 \sqcup \forall R.D_1$, $Q_2 \sqcup \forall R.\neg D_2$, and $Q_1 \sqcup \neg Q_2$, which are all Horn.

We formalize these considerations in the following definition. Intuitively, to produce a clausal form of $\forall x : \pi_y(C, x)$, each nonatomic subconcept at a position $p$ in $C$ is replaced with $\alpha$ or $\neg\alpha$, depending on the polarity of $C|_p$ in $C$, where $\alpha$ requires the same number of positive literals as $C|_p$.

**Definition 8.2.2.** *A* Horn-compatible *structural transformation is identical to the one from Definition 5.3.1, with the following difference in the definition of* $\mathsf{Def}(C)$*:*

- *If* $\Lambda(C) = \emptyset$*, then* $\mathsf{Def}(C) = \{C\}$*;*

- *Otherwise, choose* $p \in \Lambda(C)$ *and let* $\mathsf{Def}(C)$ *be as follows, where* $\alpha = Q$ *if* $\mathsf{pl}(C,p) > 0$*, and* $\alpha = \neg Q$ *if* $\mathsf{pl}(C,p) = 0$*, for* $Q$ *a new concept and* $\neg(\neg Q) = Q$*:*

$$\mathsf{Def}(C) = \begin{cases} \{\neg\alpha \sqcup C|_p\} \cup \mathsf{Def}(C[\alpha]_p) & \text{if } p \in \Lambda(C) \text{ and } \mathsf{pol}(C,p) = 1 \\ \{\neg\alpha \sqcup (\neg C)|_p\} \cup \mathsf{Def}(C[\neg\alpha]_p) & \text{if } p \in \Lambda(C) \text{ and } \mathsf{pol}(C,p) = -1 \end{cases}$$

In the rest of this section, we assume that $\Xi(KB)$ is computed using the Horn-compatible structural transformation. It is obvious that Lemma 5.3.2 holds in this case as well; that is, $KB$ and $\Xi(KB)$ are $\mathbf{D}$-equisatisfiable. The only difference to Definition 5.3.1 is that a concept at a position $p$ in $C$ can be replaced with $\neg Q$ instead of with $Q$, depending on $\mathsf{pol}(C,p)$ and $\mathsf{pl}(C,p)$. This obviously does not affect the correctness of the transformation.

We now show that applying the Horn-compatible structural transformation to Horn concepts produces Horn clauses.

**Lemma 8.2.3.** *For a Horn-*$\mathcal{SHIQ}(\mathbf{D})$ *knowledge base* $KB$*, each closure from* $\Xi(KB)$ *contains at most one positive literal.*

*Proof.* We first show the following property (*): for a Horn concept $C$, all concepts in $\mathsf{Def}(C)$ are Horn concepts. The proof is by induction on the application of the operator $\mathsf{Def}$. The induction base for $\Lambda(C) = \emptyset$ is obvious. Consider an application of $\mathsf{Def}(C)$, where $C$ is a Horn concept and $p$ is a position in $C$ such that $C|_p$ is not a literal concept, and for each position $q$ below $p$, $C|_q$ is a literal concept. Observe that, in all cases, we have $\mathsf{pl}^+(\alpha) = \mathsf{pl}(C,p)$ and $\mathsf{pl}^+(\neg\alpha) = 1 - \mathsf{pl}(C,p)$. Depending on $\mathsf{pol}(C,p)$, we consider two cases:

- For $\mathsf{pol}(C,p) = 1$, we have $\mathsf{pl}^+(\neg\alpha \sqcup C|_p) = \mathsf{pl}^+(\neg\alpha) + \mathsf{pl}^+(C|_p) = \mathsf{pl}^+(\neg\alpha) + \mathsf{pl}(C,p) = 1$. Furthermore, since $\mathsf{pl}(C,p) = \mathsf{pl}(C[\alpha]_p, p)$, $C[\alpha]_p$ is a Horn concept.
- For $\mathsf{pol}(C,p) = -1$, we have $\mathsf{pl}^+(\neg\alpha \sqcup (\neg C)|_p) = \mathsf{pl}^+(\neg\alpha) + \mathsf{pl}^-(C|_p) = \mathsf{pl}^+(\neg\alpha) + \mathsf{pl}(C,p) = 1$. Furthermore, since $\mathsf{pl}(C,p) = \mathsf{pl}(C[\neg\alpha]_p, p)$, $C[\neg\alpha]_p$ is a Horn concept.

Hence, each application of the operator $\mathsf{Def}$ decomposes a Horn concept $C$ into two simpler Horn concepts, so (*) holds. Furthermore, each immediate subconcept of $C|_p$ or $(\neg C)|_p$ is a literal concept.

For $D \in \mathsf{Def}(C)$, by definition of $\pi$ it is easy to see that $\mathsf{pl}^+(D)$ gives the maximal number of positive literals occurring in a closure from $\mathsf{Cls}(\forall x : \pi_y(D,x))$. Thus, if $C$ is a Horn concept, all closures from $\mathsf{Cls}(C)$ have at most one positive literal. All closures obtained by translating RBox and ABox axioms of $\Omega(KB)$ obviously contain at most one positive literal, so the claim of the lemma follows. □

We now show that $\mathcal{BS}_{DL}^{\mathbf{D},+}$, when applied to Horn premises, produces only Horn conclusions.

**Lemma 8.2.4.** *If all premises of an inference by $\mathcal{BS}_{DL}^{\mathbf{D},+}$ contain at most one positive literal, the inference conclusions contain at most one positive literal as well.*

*Proof.* In ordered hyperresolution and positive or negative superposition, each side premise participates in an inference on the positive literal, which does not occur in the conclusion. Hence, the number of positive literals in the conclusion is equal to the number of positive literals in the main premise. Furthermore, reflexivity resolution only reduces the number of negative literals in a closure, and equality factoring is never applicable to a closure with only one positive literal. All side premises of a concrete domain resolution inference participate in the inference on the positive literals, so the conclusion contains only negative literals. Finally, a closure participating in a decomposition inference contains the single positive literal $R(t, f(t))$, so both resulting closures have exactly one positive literal. □

The following corollary is a direct consequence of Lemmas 8.2.3 and 8.2.4:

**Corollary 8.2.5.** *If KB is a Horn $\mathcal{ALCHIQ}(\mathbf{D})$ knowledge base, then $\mathsf{DD}(KB)$ is a Horn datalog program.*

We are now ready to show the main result of this section.

**Lemma 8.2.6** (Membership)**.** *For an extensionally reduced Horn-$\mathcal{SHIQ}(\mathbf{D})$ knowledge base KB, data complexity of checking $\mathbf{D}$-satisfiability of KB is in* P*, assuming a bound on the arity of concrete predicates, and that $\mathbf{D}$-satisfiability of finite conjunctions over $\Phi_{\mathbf{D}}$ can be decided in polynomial time,*

*Proof.* Since $KB$ is $\mathbf{D}$-satisfiable if and only if $\mathsf{DD}(KB)$ is $\mathbf{D}$-satisfiable, data complexity of checking $\mathbf{D}$-satisfiability of $KB$ is bounded by the data complexity of checking $\mathbf{D}$-satisfiability of a Horn program $\mathsf{DD}(KB)$. The program $\mathsf{DD}(KB)$ can be $\mathsf{c}$-factored in time linear in $|KB_{\mathcal{A}}|$, so it suffices to show that the result is $\mathbf{d}$-satisfiable. Satisfiability of datalog programs without concrete predicates can be decided using bottom-up fixpoint saturation in polynomial time [145, 78, 35]. If $\mathsf{DD}(KB)$ contains concrete predicates, $\mathbf{d}$-satisfiability can be decided by first saturating $\mathsf{DD}(KB)$ (which can also be performed in time polynomial in the number of facts), followed by checking the $\mathbf{D}$-satisfiability of derived facts with concrete literals (which can be performed in polynomial time by the assumption). As in Lemma 8.1.2, $|\mathsf{DD}(KB)|$ is polynomial in $|KB_{\mathcal{A}}|$, so the claim of this lemma follows. □

**Lemma 8.2.7** (Hardness)**.** *For a Horn $\mathcal{ALC}$ knowledge base KB, instance checking w.r.t. KB is* P*-hard in $|KB_{\mathcal{A}}|$.*

*Proof.* The proof is implemented by the reduction from the well-known Boolean circuit value problem [104]. A *Boolean circuit* $C$ is a graph $(G, \delta, E)$ defined as follows:

- $G = \{\gamma_1, ..., \gamma_n\}$ is the set of nodes, also called *gates*;
- $\delta : G \to \{T, F, \wedge, \vee, \neg\}$ is a function assigning a label to each gate;

- $E \subseteq G \times G$ is the set of edges such that $(i)$ $E$ is acyclic—that is, $i < j$ for each $(\gamma_i, \gamma_j) \in E$; and $(ii)$ the in-degree of gates labeled with $T$ or $F$ is zero, of gates labeled with $\neg$ is one, and of gates labeled with $\wedge$ or $\vee$ is two.

A valuation $\mu : G \to \{T, F\}$ over gates of $C$ is defined inductively as follows (since $E$ is acyclic, the induction is well-founded):

- If $\delta(\gamma) \in \{T, F\}$, then $\mu(\gamma) = \delta(\gamma)$;
- If $\delta(\gamma) = \neg$ and $(\gamma_1, \gamma) \in E$, then $\mu(\gamma) = T$ if and only if $\mu(\gamma_1) = F$;
- If $\delta(\gamma) = \wedge$, $(\gamma_1, \gamma) \in E$, and $(\gamma_2, \gamma) \in E$, then $\mu(\gamma) = T$ if and only if $\mu(\gamma_1) = T$ and $\mu(\gamma_2) = T$;
- If $\delta(\gamma) = \vee$, $(\gamma_1, \gamma) \in E$, and $(\gamma_2, \gamma) \in E$, then $\mu(\gamma) = T$ if and only if $\mu(\gamma_1) = T$ or $\mu(\gamma_2) = T$.

The *value* of $C$ is defined as $\mu(C) = \mu(\gamma_n)$. For an arbitrary circuit $C$, checking if $\mu(C) = T$ is P-complete [104].

For a circuit $C$, we construct the knowledge base $KB_C$ in which each gate $\gamma_i$ corresponds to the individual $\gamma_i$. We convert the graph structure of $C$ into ABox assertions as follows:

- If $\delta(\gamma) = \neg$ and $(\gamma_1, \gamma) \in E$, we add the ABox assertion $not(\gamma, \gamma_1)$;
- If $\delta(\gamma) = \wedge$, $(\gamma_1, \gamma) \in E$, and $(\gamma_2, \gamma) \in E$, we add the ABox assertions $and_1(\gamma, \gamma_1)$ and $and_2(\gamma, \gamma_2)$;
- If $\delta(\gamma) = \vee$, $(\gamma_1, \gamma) \in E$, and $(\gamma_2, \gamma) \in E$, we add the ABox assertions $or_1(\gamma, \gamma_1)$ and $or_2(\gamma, \gamma_2)$;
- If $\delta(\gamma) = T$, we add the ABox assertion $T(\gamma)$;
- If $\delta(\gamma) = F$, we add the ABox assertion $F(\gamma)$.

Furthermore, the TBox of $KB_C$ contains the following axioms:

$$\begin{array}{rr}
\exists not.T \sqsubseteq F & T \sqcap F \sqsubseteq \bot \\
\exists not.F \sqsubseteq T & \\
\exists and_1.T \sqcap \exists and_2.T \sqsubseteq T & \exists or_1.T \sqcap \exists or_2.T \sqsubseteq T \\
\exists and_1.T \sqcap \exists and_2.F \sqsubseteq F & \exists or_1.T \sqcap \exists or_2.F \sqsubseteq T \\
\exists and_1.F \sqcap \exists and_2.T \sqsubseteq F & \exists or_1.F \sqcap \exists or_2.T \sqsubseteq T \\
\exists and_1.F \sqcap \exists and_2.F \sqsubseteq F & \exists or_1.F \sqcap \exists or_2.F \sqsubseteq F
\end{array}$$

It is now easy to see that $\mu(C) = T$ if and only if $KB_C \models T(\gamma_n)$. Namely, the TBox axioms ensure that that concept membership is propagated through the circuit

according to the standard semantics of propositional connectives. Hence, for each gate $\gamma$, $\mu(\gamma) = T$ if and only if $KB_C \models T(\gamma)$, and $\mu(\gamma) = F$ if and only if $KB_C \models F(\gamma)$.[2]

Now the claim of the lemma follows because the size of the TBox of $KB_C$ is constant, the size of the ABox of $KB_C$ is linear in the size of $C$, and $KB_C$ is a Horn knowledge base. □

The following theorem is an immediate consequence of these two lemmata.

**Theorem 8.2.8.** *Let $KB$ be an extensionally reduced Horn knowledge base expressed in any logic between $\mathcal{ALC}$ and $\mathcal{SHIQ}(\mathbf{D})$. Assuming a polynomial oracle for reasoning with a concrete domain and a bound on the arity of concrete domain predicates, the problems of checking $KB$ ($\mathbf{D}$-)(un)satisfiability, and checking if $KB \models_{(\mathbf{D})} \alpha$ are data complete for* P*, for $\alpha$ of the form $(\neg)R(a,b)$ or of the form $(\neg)C(a)$ with $C$ a Horn concept of bounded size.*

## 8.3 Discussion

To better understand the results from the previous two sections, we contrast them with the well-known results for (disjunctive) datalog [35]. Since datalog has been successfully applied in practice, this analysis gives interesting insights into the practical applicability of description logics.

Note that the complexities of datalog variants and of corresponding $\mathcal{SHIQ}$ fragments seem to coincide. Without disjunctions, a $\mathcal{SHIQ}$ knowledge base and a datalog program always have at most one model, which can be computed in polynomial time. With disjunctions, several models are possible, and this must be dealt with using reasoning by case. Intuitively, one needs to guess a model, which increases data complexity to NP.

The key difference between datalog and description logics is revealed by considering the effects that various parameters have on the complexity. For a datalog program $P$ and a ground atom $\alpha$, checking whether $P \models \alpha$ can be performed in time $\mathcal{O}(|P|^v)$, where $v$ is the maximal number of distinct variables in a rule of $P$ [146]. Namely, the problem can be solved by grounding $P$—that is, by replacing in each rule of $P$ all variables with individuals from $P$ in all possible ways. The size of the grounding is bounded by $|P|^v$, and propositional Horn logic is P-complete, which implies $\mathcal{O}(|P|^v)$ complexity. In general, $v$ is linear in $|P|$, so the size of the grounding is exponential; thus, the combined complexity of datalog coincides with the combined complexity of $\mathcal{SHIQ}$. However, in practical applications, $v$ is usually small, so it makes sense to assume it is bounded; then, datalog actually exhibits polynomial behavior.

An intuitive analogous limitation for DLs might be to restrict the size of concepts in axioms. However, this is not an adequate limitation. Namely, DLs are closely related

[2]Note that the structure of $KB_C$ ensures that each gate is a member of either $T$ or $F$ concept, even though the TBox does not contain an axiom $\top \sqsubseteq T \sqcup F$. Furthermore, including such an axiom into $KB_C$ would invalidate the proof, because this would make $KB_C$ a non-Horn knowledge base.

to the two-variable fragment of first-order logic [21]: $\mathcal{ALC}$ concepts correspond to first-order formulae with only two variables, regardless of nesting. By limiting the numbers occurring in number restrictions to $n$, the number of variables in $\mathcal{SHIQ}$ axioms is limited to $n + 2$. Axioms with complex concepts can be polynomially reduced to axioms with atomic concepts using structural transformation.

We summarize our observations as follows: assuming a bound on the axiom length, but not on the number of axioms, satisfiability checking in datalog is (nondeterministically) polynomial, but in DLs it is exponential. This is so because DLs such as $\mathcal{ALC}$ provide the existential quantifier and general inclusion axioms, which can be used to succinctly encode models with paths of exponential length. The saturation step eliminates the function symbols, but it also incurs an exponential blowup in the program size to account for such paths. Hence, although combined complexity of both datalog and DLs is exponential, the reasons for this are different.

In [4, Chapter 5], two sources of complexity in DLs have been identified: OR-branching caused by the existence of numerous possible models, and AND-branching caused by the existence of paths within a model. Our results show that OR-branching is not as difficult as AND-branching: the former increases the complexity to NP, whereas the latter increases the complexity to ExpTime.

## 8.4 Related Work

Data complexity of reasoning in description logics has so far been rarely considered. The only work we are aware of is [126, 125], where the complexity bounds of instance checking for certain description logics were considered. In particular, it was shown that combined complexity of instance checking for languages with polynomial subsumption algorithm, such as $\mathcal{AL}$ and $\mathcal{ALN}$, is also polynomial. For $\mathcal{ALE}$ it was shown that instance checking is co-NP-hard in the size of data; however, $\Pi_2^{\mathrm{P}}$ is given as the upper bound. Furthermore, it was shown that the combined complexity of instance checking in $\mathcal{ALE}$ is PSpace-complete. Finally, for $\mathcal{ALR}$, it was shown that the combined complexity of instance checking is NP-complete. It was pointed out that the additional source of complexity in $\mathcal{ALE}$ over $\mathcal{ALR}$ arises from qualified existential quantification.

Whereas the hardness proof for $\mathcal{ALE}$ provides a lower bound for data complexity of satisfiability, the upper bound does not follow from [126], since none of the logics considered is as expressive as $\mathcal{ALC}$ or $\mathcal{SHIQ}(\mathbf{D})$. In particular, no considered logic allows for general inclusion axioms, which are known to have a significant impact on the complexity. Furthermore, the complexity of reasoning in [126] is measured in the size of the ABox that is allowed to contain complex concept expressions. Our work addresses the logic with general inclusion axioms, but restricts the ABox to contain only literal concepts. Hence, our results measure the complexity in the number of simple facts, without any terminological knowledge in the ABox.

# Part III

# Extensions

# Chapter 9

# Integrating Description Logics with Rules

Although $\mathcal{SHIQ}(\mathbf{D})$ is very expressive, it is a *decidable* fragment of first-order logic, and thus cannot express arbitrary axioms. In fact, it can express only axioms of a certain tree-structure [57]. Decidable rule-based formalisms, such as function-free Horn rules,[1] do not have this restriction, but lack some of the expressive power of $\mathcal{SHIQ}(\mathbf{D})$, since they are restricted to universal quantification and, in their basic form, provide no negation. To overcome the limitations of both approaches, description logics were extended with rules [69], but this extension is undecidable [69]. Intuitively, the undecidability arises because adding rules to $\mathcal{SHIQ}(\mathbf{D})$ causes the loss of any form of *tree model property* [147]. In a logic with such a property, every satisfiable knowledge base has a model of a certain tree-shaped form, so, to decide satisfiability, it suffices to search only for such a model. For most DLs, it is possible to ensure termination of such a search.

It is natural to ask what kind of rules can be added to $\mathcal{SHIQ}(\mathbf{D})$ while preserving decidability. This follows a classic line of research in knowledge representation of investigating the trade-off between expressivity and complexity, and providing different formalisms with varying expressive power and complexity. This not only provides for better understanding of the causes for the undecidability of the full combination, but also enables a more detailed analysis of the complexity and, ultimately, the design of specialized reasoning procedures. Applications that do not require the expressive power of the full combination can use such procedures, and rely on the known upper time and space bounds required to return a correct answer. Finally, in the last decade, it turned out that these specialized decision procedures are amenable to optimizations, thus achieving surprisingly good performance in practice even for logics with high worst-case complexity [4, Chapter 9].

In this chapter, we present a decidable combination of $\mathcal{SHIQ}(\mathbf{D})$ with rules, where decidability is achieved by requiring the rules to be *DL-safe*: concepts (roles) are

[1] Throughout this chapter, we use "rules" and "clauses" as synonyms, following [69].

Table 9.1: Example Knowledge Base

| | |
|---|---|
| $Person(Peter)$ | Peter is a person. |
| $Person \sqsubseteq \exists father.Person$ | Each person has a father who is a person. |
| $\exists father.(\exists father.Person) \sqsubseteq Grandchild$ | Things having a father of a father who is a person are grandchildren. |

allowed to occur in both rule bodies and heads as unary (binary) predicates in atoms, but each variable in a rule is required to occur in a body literal whose predicate is neither a concept nor a role. Intuitively, this restriction makes the logic decidable because the rules are applicable only to individuals explicitly introduced in the ABox. Our formalism is a generalization of existing hybrid approaches presented in [84, 39], and a special case of approaches presented in [123, 69]. We discuss the expressive power and the limitations of our approach in a nontrivial example. Moreover, we show that query answering with DL-safe rules can be performed by simply appending the rules to a program obtained by the reduction from Chapter 7.

## 9.1 Reasons for Undecidability of $\mathcal{SHIQ}(\mathbf{D})$ with Rules

An approach for integrating an OWL-DL knowledge base $KB$ with a datalog program $P$ was presented in [69]. The integration is achieved by assuming $N_C \cup N_{R_a} \cup N_{R_c} \subseteq N_P$, and by interpreting the resulting hybrid knowledge base under standard first-order semantics (that is, the rules in $P$ are interpreted as clauses). In other words, concepts and roles can be used in rules as unary and binary predicates, respectively. It was shown that checking satisfiability of such a hybrid knowledge base is undecidable; as a consequence, subsumption and query answering w.r.t. hybrid knowledge bases are also undecidable. Investigating the proof from [69] and [84] more closely, we note that the undecidability is caused by the interaction between some very basic features of description logics and rules. In this section, we try to give an intuitive explanation of this result and its consequences.

Consider the simple knowledge base $KB$ from Table 9.1. This knowledge base implies the existence of an infinite chain of fathers: since *Peter* must have a father, there is some $x_1$ who is a *Person*. In turn, $x_1$ must have some father $x_2$, who must be a *Person*, and so on. An infinite model with such a chain is shown in Figure 9.1, upper part (a). Observe that *Peter* is a grandchild, since he has a father of a father, who is a person.

Let us now check whether $KB \models Grandchild(Jane)$; this is the case if and only if $KB \cup \{\neg Grandchild(Jane)\}$ is unsatisfiable. We can perform this task by trying to build a model; if we fail, then we conclude that $KB \cup \{\neg Grandchild(Jane)\}$ is unsatisfiable. However, we have a problem: starting from *Peter*, a naïve approach will expand the chain of Peter's fathers indefinitely, and will therefore not terminate.

This very simple example intuitively shows that we have to be careful if we want to ensure termination of a model building algorithm. For many DLs, termination can be ensured without losing completeness because we can restrict our attention to certain "nice" models. For numerous DLs, we can consider only *tree models*—that is, models in which the underlying relational structure forms a tree [147]. This is so because every satisfiable knowledge base has such a tree model (to be precise, for certain logics we consider tree-like abstractions of possibly nontree models, but this distinction is not relevant here). Even if such a tree model is infinite, we can *wind* this infinite tree model into a finite one. In our example, since $KB$ does not require each father in the chain to be distinct (that is, there is no axiom requiring the role *father* to be acyclic), the model in Figure 9.1, lower part (b), is the result of winding an infinite tree into a "nice," finite model. Due to their regular structure, such windings of tree models can be easily constructed in an automated way. To understand why every satisfiable $\mathcal{SHIQ}(\mathbf{D})$ knowledge base has a tree model [73], consider the mapping $\pi$ from Tables 3.1 and 3.4 more closely (we ignore some technicalities caused by transitive roles): in all formulae obtained by transforming the result of $\pi$ into prenex normal form, variables are connected by roles only in a tree-like manner, as shown in the following example:

$$\begin{array}{l} \exists S.(\exists R.C \sqcap \exists R.D) \sqsubseteq Q \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \Rightarrow \\ \forall x : \{[\exists y : S(x,y) \wedge (\exists x : R(y,x) \wedge C(x)) \wedge (\exists x : R(y,x) \wedge D(x))] \rightarrow Q(x)\} \Rightarrow \\ \forall x, x_1, x_2, x_3 : \{S(x,x_1) \wedge R(x_1,x_2) \wedge C(x_2) \wedge R(x_1,x_3) \wedge D(x_3) \rightarrow Q(x)\} \end{array}$$

Let us contrast these observations with the kind of reasoning required for function-free Horn rules. In such rules, all variables are universally quantified; that is, there are no existentially quantified variables in rule consequents. Hence, we never have to infer the existence of objects not enumerated in the Herbrand universe. Thus, reasoning algorithms can consider only individuals that are explicitly introduced and are given a name in the knowledge base. Reasoning can be performed by *grounding* the rules—that is, by replacing the variables in the rules with all individuals from the knowledge base in all possible ways. Through grounding, first-order reasoning becomes propositional, since a ground rule is essentially equivalent to a propositional clause. For a finite program, the number of ground rules is also finite, and satisfiability of a set of propositional clauses is decidable. Hence, non-tree-like rules are allowed to enforce arbitrary, but finite, nontree models, and not only "nice" models.

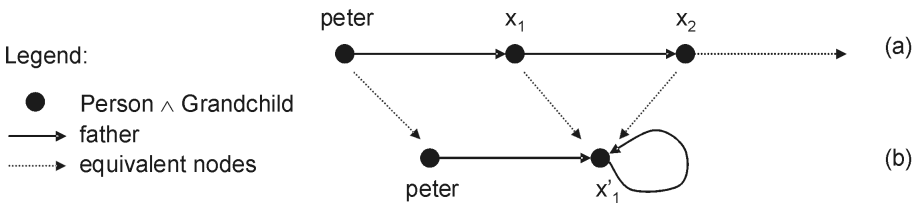



Figure 9.1: Two Similar Models

Now let us see what happens if we extend $\mathcal{SHIQ}(\mathbf{D})$ with function-free Horn rules. Then, we combine a logic that can be decided by restricting our attention to "nice" models (but with possibly infinitely many individuals whose existence is implied by a knowledge base) with a logic that can be decided by restricting our attention to known individuals (but with arbitrary relations between them). Unsurprisingly, this and similar combinations are undecidable [84, 69].

## 9.2 Combining Description Logics and Rules

We now formalize the interface between $\mathcal{SHIQ}(\mathbf{D})$ and rules.

**Definition 9.2.1** (DL-Rules)**.** *Let $KB$ be a $\mathcal{SHIQ}(\mathbf{D})$ knowledge base. For $s$ and $t$ constants or variables, a* DL-atom *is an atom of the form $A(s)$, where $A$ is an atomic concept in $KB$, of the form $R(s,t)$ where $R$ is simple (abstract or concrete) role in $KB$, or of the form $s \approx t$. A* non-DL-atom *is an atom with a predicate other than $\approx$, an atomic concept in $KB$, or a role in $KB$. A (disjunctive)* DL-rule *is a (disjunctive) rule with DL- and non-DL-atoms in the head and the body. A (disjunctive) DL-program $P$ is a set of (disjunctive) DL-rules. A* combined knowledge base *is a pair $(KB, P)$.*

*We define the translation operator $\pi$ on rules to interpret them as first-order clauses ($\mathbf{x}$ is the vector of all free variables of the rule):*

$$\pi(A_1 \vee ... \vee A_n \leftarrow B_1, ..., B_m) = \forall \mathbf{x} : \{A_1 \vee ... \vee A_n \vee \neg B_1 \vee ... \vee \neg B_m\}$$

*For a DL-program $P$, we define $\pi(P) = \bigwedge_{r \in P} \pi(r)$, and, for a combined knowledge base $(KB, P)$, we define $\pi((KB, P)) = \pi(KB) \wedge \pi(P)$. The main inferences for combined knowledge bases are defined as follows:*

- Satisfiability checking: *$(KB, P)$ is* satisfiable *if and only if $\pi((KB, P))$ is $\mathbf{D}$-satisfiable.*
- Query answering: *A ground atom $\alpha$ is an* answer *of $(KB, P)$, denoted with $(KB, P) \models \alpha$, if and only if $\pi((KB, P)) \models_{\mathbf{D}} \pi(\alpha)$.*

A few remarks regarding Definition 9.2.1 are in order.

**Minimal vs. First-order Models.** Rules are usually interpreted under minimal model semantics. In contrast, the semantics of DL-rules is classical, in the sense that it considers arbitrary models. An in-depth comparison between the two types of semantics was presented in Subsection 4.8.2.

**Transitive Roles.** It is straightforward to extend Definition 9.2.1 to allow DL-atoms to contain complex roles. However, our algorithms deal with transitivity axioms by encoding a $\mathcal{SHIQ}(\mathbf{D})$ knowledge base $KB$ into a $\mathbf{D}$-equisatisfiable $\mathcal{ALCHIQ}(\mathbf{D})$ knowledge base $\Omega(KB)$. As mentioned in Section 5.2, this transformation does not preserve entailment of ground complex role atoms. Therefore, we prohibit the usage of complex roles in rules (such roles can still be used in $KB$).

**(In)equality Literals.** Without loss of generality, we allow only positive equality literals in DL-rules. Namely, DL-rules can be disjunctive and are interpreted under standard first-order semantics, so a negative equality literal in the body can be moved as a positive literal into the head. For example, the following rule defines *Commuter* as a person who lives and works at different places:

$$Commuter(x) \leftarrow livesAt(x, y), worksAt(x, y), x \not\approx y$$

By standard properties of first-order logic, this rule is equivalent to the following one:

$$Commuter(x) \vee x \approx y \leftarrow livesAt(x, y), worksAt(x, y)$$

By allowing only positive equality literals in rules, we make Definition 9.2.1 consistent with the definition of disjunctive datalog from Section 2.7, which allows only for positive atoms in rule bodies.

Note also that positive equality literals in the rule body do not increase the expressivity of the formalism. Namely, for a first-order formula $\varphi$, a variable $x$, and a term $t$ that does not contain $x$ as a proper subterm, the formulae $\forall x : (x \approx t \rightarrow \varphi)$ and $\varphi\{x \mapsto t\}$ are equisatisfiable [99]. However, positive equality literals in the rule heads do increase the expressivity, as they allow deriving two objects to be equal. In order not to make Definition 9.2.1 too technical, we do not distinguish these two cases.

**Relationship with Existing Formalisms.** Definition 9.2.1 yields a formalism compatible with the ones from [69, 84, 123]. The main differences from [69] are that DL-rules can additionally contain non-DL-atoms; furthermore, DL-atoms cannot contain inequality, and they can contain only simple roles and atomic concepts. The latter is a technical assumption and is not really a restriction: for a complex concept $C$, one can always introduce a new atomic concept $A_C$, add the axiom $A_C \equiv C$ to the TBox, and use $A_C$ in the rule. This transformation is obviously linear in the size of $P$.

**Decidability.** Since the formalism is compatible with [69], reasoning with combined knowledge bases is undecidable. To achieve decidability, we introduce the notion of DL-safety in the following section.

## 9.3 DL-Safety Restriction

A possible way to obtain a decidable logic is to require DL-rules to be DL-safe.

**Definition 9.3.1** (DL-Safe Rules)**.** *A (disjunctive) DL-rule $r$ is* DL-safe *if each variable occurring in $r$ also occurs in a non-DL-atom in the body of $r$. A (disjunctive) DL-program $P$ is DL-safe if all its rules are DL-safe.*

DL-safety is similar to safety in datalog. In a safe rule, each variable occurs in a positive atom in the body, and can therefore be bound only to constants explicitly

present in the database. Similarly, DL-safety ensures that each variable is bound only to individuals explicitly introduced in the ABox. For example, if *Person*, *livesAt*, and *worksAt* are concepts and roles from $KB$, the following rule is not DL-safe, because both $x$ and $y$ occur in DL-atoms, but not in an atom with a predicate outside of $KB$:

$$Homeworker(x) \leftarrow Person(x), livesAt(x, y), worksAt(x, y)$$

The previous rule can be made DL-safe by adding special non-DL-atoms $\mathcal{O}(x)$, $\mathcal{O}(y)$, and $\mathcal{O}(z)$ to the body of the rule, and by adding a fact $\mathcal{O}(a)$ for each individual $a$ occurring in $KB$ and $P$. Thus, the previous rule is transformed as follows:

$$Homeworker(x) \leftarrow Person(x), livesAt(x, y), worksAt(x, y), \mathcal{O}(x), \mathcal{O}(y), \mathcal{O}(z)$$

## 9.4 Expressivity of DL-Safe Rules

To achieve decidability, we do not restrict the component languages; rather, we combine full $\mathcal{SHIQ}(\mathbf{D})$ with function-free Horn rules, and thus extend both formalisms. However, DL-safety restricts the interaction between the component languages to involve only individuals explicitly introduced in the ABox.

To illustrate the expressive power of DL-safe rules, consider the combined knowledge base $(KB, P)$, containing DL axioms and rules from Table 9.2. We use a rule to define the only non-DL-predicate *BadChild* as a grandchild who hates some of his siblings (or himself). Note that this rule involves relations forming a triangle between two siblings and a parent, and thus cannot be expressed in $\mathcal{SHIQ}(\mathbf{D})$; furthermore, the rule is not DL-safe.

Now consider the first group of ABox facts. Since *Cain* is a *Person*, as shown in Section 9.1, one can infer that *Cain* is a *Grandchild*. Since *Cain* and *Abel* are children of *Adam*, and *Cain hates Abel*, we derive that *Cain* is a *BadChild*.

Similarly, since *Romulus* and *Remus* are persons, they have a father. Due to the second rule, the father of *Romulus* and *Remus* must be the same. Now *Romulus hates Remus*, so *Romulus* is a *BadChild* as well. We are able to derive this without knowing exactly who the father of *Romulus* and *Remus* is.[2]

Consider now the DL-safe rule defining *BadChild*′: since the father of *Cain* and *Abel* is known by name—that is, *Adam* is in the ABox—, the literal $\mathcal{O}(y)$ from the rule for *BadChild*′ can be matched to $\mathcal{O}(Adam)$, allowing us to conclude that *Cain* is a *BadChild*′. In contrast, the father of *Romulus* and *Remus* is not known in the ABox. Hence, in the DL-safe version of the second rule, $\mathcal{O}(x)$ and $\mathcal{O}(y)$ cannot be matched to the father's name, so the rule does not derive that the fathers of *Romulus* and *Remus* are the same. Similarly, in the rule defining *BadChild*′, the literal $\mathcal{O}(y)$ cannot be matched to the father's name, so we cannot derive that *Romulus* is a *BadChild*′.

This may seem confusing. However, DL-safe rules do have a natural reading: just append the phrase "where the identity of all objects is known" to the intuitive meaning

[2] Actually, the father of Romulus and Remus is the god Mars, but we assume that this is not known to the modeler of $KB$. Still, the father's existence is certain, which is reflected by $KB$.

Table 9.2: Example with DL-Safe Rules

| | |
|---|---|
| $Person \sqsubseteq \exists father.Person$ | Each person has a father who is a person. |
| $\exists father.(\exists father.Person) \sqsubseteq Grandchild$ | Things having a father of a father who is a person are grandchildren. |
| $father \sqsubseteq parent$ | Fatherhood is a kind of parenthood. |
| $BadChild(x) \leftarrow Grandchild(x),$ $parent(x,y), parent(z,y), hates(x,z)$ | A bad child is a grandchild who hates one of his siblings. |
| $BadChild'(x) \leftarrow Grandchild(x),$ $parent(x,y), parent(z,y), hates(x,z),$ $\mathcal{O}(x), \mathcal{O}(y), \mathcal{O}(z)$ | DL-safe version of a bad child. |
| $Person(Cain)$ | Cain is a person. |
| $father(Cain, Adam)$ | Cain's father is Adam. |
| $father(Abel, Adam)$ | Abel's father is Adam. |
| $hates(Cain, Abel)$ | Cain hates Abel. |
| $Person(Romulus)$ | Romulus is a person. |
| $Person(Remus)$ | Remus is a person. |
| $x \approx y \leftarrow father(Romulus, x), father(Remus, y)$ | Romulus and Remus have the same father. |
| $x \approx y \leftarrow father(Romulus, x), father(Remus, y),$ $\mathcal{O}(x), \mathcal{O}(y)$ | DL-safe version. |
| $hates(Romulus, Remus)$ | Romulus hates Remus. |
| $Child(x) \leftarrow GoodChild(x), \mathcal{O}(x)$ | Good children are children. |
| $Child(x) \leftarrow BadChild'(x), \mathcal{O}(x)$ | Bad children are children. |
| $(GoodChild \sqcup BadChild')(Oedipus)$ | Oedipus is a good or a bad child. |
| $\mathcal{O}(\alpha)$ for each explicitly named individual $\alpha$ | Enumeration of all ABox individuals. |

of the rule. For example, the rule defining *BadChild′* should be read as "A *BadChild′* is a *known* grandchild whose parent is *known*, and who hates one of his *known* siblings."

Combining description logics with DL-safe rules increases the expressivity of both components. Namely, a $\mathcal{SHIQ}(\mathbf{D})$ knowledge base cannot imply that *Cain* is a *BadChild′*, because the rule expressing a triangular relationship cannot be expressed in $\mathcal{SHIQ}(\mathbf{D})$. Similarly, function-free Horn rules cannot imply this either: we know that *Cain* has a grandfather because *Cain* is a person, but we do not know who he is. Hence, we need the existential quantifier to infer the existence of ancestors, and then to infer that *Cain* is a *Grandchild*.

Note that it is incorrect to compute all consequences of the DL component first, and then to apply the rules to these consequences. Consider the *KB* part about *Oedipus*: he is a *GoodChild* or a *BadChild′*, but we do not know exactly what is true. Either way, one of the rules derives that *Oedipus* is a *Child*, so $(KB, P) \models Child(Oedipus)$. This would not be derived by applying the rules for *Child* to the consequences of *KB*, since $KB \not\models GoodChild(Oedipus)$ and $KB \not\models BadChild'(Oedipus)$.

## 9.5 Query Answering for DL-Safe Rules

We now show that reasoning in $(KB, P)$ can be performed by simply appending $P$ to $\mathsf{DD}(KB)$. To do that, we extend several lemmata used in the proof of Theorem 7.4.2.

For a $\mathcal{SHIQ}(\mathbf{D})$ knowledge base $KB$, the first step in query answering is to eliminate transitivity axioms by encoding $KB$ into an $\mathbf{D}$-equisatisfiable $\mathcal{ALCHIQ}(\mathbf{D})$ knowledge base $\Omega(KB)$, as explained in Section 5.2. Observe that Definition 9.3.1 allows only simple roles to occur in DL-atoms, and that, as discussed in Section 5.2, this encoding preserves entailment of such atoms. Hence, for $P$ a DL-safe program, $(KB, P)$ and $(\Omega(KB), P)$ are equisatisfiable. Hence, in the rest of this section we assume without loss of generality that $KB$ is an $\mathcal{ALCHIQ}(\mathbf{D})$ knowledge base.

Let $P^{\mathsf{c}}$ be a $\mathsf{c}$-factor of $\pi(P)$. Without loss of generality, we can assume that $P^{\mathsf{c}}$ is computed by first applying $\mathsf{c}$-factoring to all negative DL-atoms, and then to non-DL-atoms in a rule. Thus, for a clause $C$ from $P^{\mathsf{c}}$, negative DL-atoms containing a concrete role occur in $C$ only in disjunctions of the form $\neg T(x, z^{\mathsf{c}}) \vee z^{\mathsf{c}} \not\approx_{\mathbf{D}} y^{\mathsf{c}}$, where $y^{\mathsf{c}}$ occurs in a non-DL-atom because of DL-safety. Furthermore, in all positive DL-atoms $T(x, y^{\mathsf{c}})$ of $C$, $y^{\mathsf{c}}$ also occurs in a negative non-DL-atom because of DL-safety.

By Lemma 6.1.6, $\mathbf{D}$-satisfiability of $\Xi(KB) \cup P$ coincides with $\mathbf{d}$-satisfiability of $\Xi(KB) \cup P^{\mathsf{c}}$, and the latter can be decided by $\mathcal{BS}_{DL}^{\mathbf{D},+}$. To obtain a decision procedure, we extend the selection function of $\mathcal{BS}_{DL}^{\mathbf{D},+}$ as follows: if a closure contains negative non-DL-atoms, then all such atoms, and nothing else are selected; otherwise, if there are no negative non-DL-atoms, the selection function is as in Definition 5.3.3—that is, it selects all negative binary literals.

We define the *extended* $\mathcal{ALCHIQ}(\mathbf{D})$*-closures* to include the closure types from Table 5.2 without conditions (*iii*)–(*vi*), the closure types from Table 6.2, and the closures corresponding to $\mathsf{c}$-factors of DL-safe rules. Furthermore, closures of type 8 are allowed to contain positive ground non-DL-atoms and negative non-DL-atoms whose arguments are either variables or constants.

**Lemma 9.5.1.** *For an* $\mathcal{ALCHIQ}(\mathbf{D})$ *knowledge base* $KB$*, saturation by* $\mathcal{BS}_{DL}^{\mathbf{D},+}$ *decides* $\mathbf{d}$*-satisfiability of* $\Xi(KB) \cup P^{\mathsf{c}}$*.*

*Proof.* All closures from $\Xi(KB) \cup P^{\mathsf{c}}$ are obviously extended $\mathcal{ALCHIQ}(\mathbf{D})$-closures. To prove the lemma, it is sufficient to show the following property (*): in a $\mathcal{BS}_{DL}^{\mathbf{D},+}$-derivation $\Xi(KB) \cup P^{\mathsf{c}} = N_0, \ldots, N_i \cup \{C\}$, the closure $C$, derived from premises in $N_i$, is either an extended $\mathcal{ALCHIQ}(\mathbf{D})$-closure, or it is redundant in $N_i$.

All ground non-DL-atoms in $\Xi(KB) \cup P^{\mathsf{c}}$ contain constants. Hence, a superposition into a ground non-DL-atom is possible only from a literal $\langle a \rangle \approx \langle b \rangle$, and in the superposition conclusion all non-DL-atoms contain constants. Consider an inference with a rule $r$. Since $r$ is DL-safe, it can participate only in a hyperresolution inference with side premises of type 8 on non-DL-literals. Furthermore, $r$ is safe, so, since all ground non-DL-atoms contain constants, hyperresolution binds all variables in a rule to constants. An exception is variables $z^{\mathsf{c}}$ in literals of the form $\neg T(x, z^{\mathsf{c}})$ with $T$ a concrete role. However, due to $\mathsf{c}$-factoring and DL-safety, such literals occur

in rules always as part of a disjunction $\neg T(x, z^{\mathsf{c}}) \vee z^{\mathsf{c}} \not\approx_{\mathbf{D}} y^{\mathsf{c}}$, where $x$ and $y^{\mathsf{c}}$ occur in non-DL-atoms, so these literals have in the hyperresolution conclusion the form $\neg T(a, z^{\mathsf{c}}) \vee z^{\mathsf{c}} \not\approx_{\mathbf{D}} b^{\mathsf{c}}$. Obviously, the conclusion is a closure of type 8. Furthermore, closures of type 8 can participate in $\mathcal{BS}_{DL}^{\mathbf{D},+}$ inferences with other closures in exactly the same way as in Lemma 5.3.6, Lemma 6.2.1, and Theorem 5.4.8, so the property (*) holds. Since $\mathcal{BS}_{DL}^{\mathbf{D},+}$ is sound and complete calculus for checking **d**-satisfiability, the claim of the lemma follows. □

The next step is to show that rules can simply be appended to the function-free version of $KB$.

**Lemma 9.5.2.** *$(KB, P)$ is unsatisfiable if and only if $\mathsf{FF}(KB) \cup P^{\mathsf{c}}$ is **d**-unsatisfiable.*

*Proof.* By Theorem 6.1.20, **d**-satisfiability of $\mathsf{FF}(KB) \cup P^{\mathsf{c}}$ can be decided by a $\mathcal{BS}_{DL}^{\mathbf{D},+}$ saturation, in which we can choose to perform all nonground inferences before all ground inferences. Since all non-DL-atoms in rules from $P^{\mathsf{c}}$ are selected and each rule by the DL-safety requirement must contain at least one such atom, the rules cannot participate in an inference with nonground closures from $\Xi(KB) \cup P^{\mathsf{c}}$. Hence, as in Lemma 7.2.1, $\Xi(KB) \cup P^{\mathsf{c}}$ is **d**-satisfiable if and only if $\Gamma = \mathsf{Sat}_{\mathsf{R}}(\Gamma_{\mathcal{TR}g}) \cup \Xi(KB_{\mathcal{A}}) \cup P^{\mathsf{c}}$ is **d**-satisfiable.

It is now straightforward to extend Lemma 7.2.4 to show that $\Gamma$ is **d**-unsatisfiable if and only if $\mathsf{FF}(KB) \cup P^{\mathsf{c}}$ is **d**-unsatisfiable. For both directions of the proof, a hyperresolution with a rule $r$ in one closure set can directly be simulated by a hyperresolution with $r$ in the other closure set. □

We now state the main result of this section:

**Theorem 9.5.3.** *Let $KB$ be an $\mathcal{ALCHIQ}(\mathbf{D})$ knowledge base, and $P$ a DL-safe disjunctive datalog program. Then, $(KB, P)$ is unsatisfiable if and only if $\mathsf{DD}(KB) \cup P$ is **D**-unsatisfiable. Furthermore, $(KB, P) \models \alpha$ if and only if $\mathsf{DD}(KB) \cup P \models_c \alpha$, where $\alpha$ is a ground DL- or a non-DL-atom.*

*Proof.* Because $\mathsf{FF}(KB) \cup P$ is **D**-satisfiable if and only $\mathsf{FF}(KB) \cup P^{\mathsf{c}}$ is **d**-satisfiable, the first claim follows from Lemma 9.5.2. For the second claim, observe that $(KB, P) \models \alpha$ if and only if $(KB \cup \{\neg\alpha\}, P)$ is unsatisfiable. The latter is the case if and only if $\mathsf{FF}(KB \cup \{\neg\alpha\}) \cup P = \mathsf{FF}(KB) \cup P \cup \{\neg\alpha\}$ is **D**-unsatisfiable, which is the case if and only if $\mathsf{DD}(KB) \cup P \models_c \alpha$. □

Query answering in $\mathsf{DD}(KB) \cup P$ can be performed using the algorithm from Section 7.6, for which we now determine the complexity. As explained in Section 7.5, $P_{\approx}$ is the axiomatization of the equality semantics for predicates occurring in a program $P$.

**Theorem 9.5.4.** *Let $KB$ be an $\mathcal{ALCHIQ}(\mathbf{D})$ knowledge base, defined over a concrete domain **D** such that **D**-satisfiability of finite conjunctions over $\Phi_{\mathbf{D}}$ can be decided in deterministic exponential time. Also, let $P$ be a DL-safe program, $\Gamma' = \mathsf{DD}(KB) \cup P$, and $\Gamma = \Gamma' \cup \Gamma'_{\approx}$. Finally, let numbers be coded in unary, and the arity of concrete*

*predicates be bounded. Then, computing all answers to a nonground query in* $(KB, P)$ *can be done by saturating a* $\mathsf{c}$*-factor of* $\Gamma$ *by* $\mathcal{R}_Q^{\mathbf{D}}$ *in time exponential in* $|KB| + |P|$ *assuming a bound on the arity of predicates in* $P$*, and in time doubly exponential in* $|KB| + |P|$ *otherwise.*

*Proof.* The number of clauses in $\Gamma'_{\approx}$ is polynomial in $|KB| + |P|$. As done in Lemma 5.3.9, to determine the complexity of the algorithm, we compute the maximal number of clauses derivable in a saturation by $\mathcal{R}_Q^{\mathbf{D}}$. Let $p$ denote the number of non-DL-predicates, $r$ the maximal arity of a predicate, $c$ the number of constants occurring in $(KB, P)$, and $b$ the maximal number of literals in a body of a rule from $P$. Under the theorem assumptions, $p$, $c$, and $b$ are linear in $|KB| + |P|$. If $r$ is not bounded, it is also linear in $|KB| + |P|$.

The number of non-DL-literals occurring in a maximal ground clause is bounded by $\ell_1 = 2pc^r$ (the factor 2 allows each literal to occur positively or negatively). If the predicate arity is bounded, then $\ell_1$ is polynomial, and if the predicate arity is unbounded, it is exponential in $|KB| + |P|$. By Lemma 5.3.9, we know that the maximal number of DL-atoms in a clause, denoted with $\ell_2$, is polynomial in $|KB|$ assuming a bound on the arity of concrete domain predicates and for unary coding of numbers. Since each ground clause can contain an arbitrary subset of these literals, the maximal number of ground clauses derived by $\mathcal{R}_Q^{\mathbf{D}}$ is bounded by $k = 2^{\ell_1+\ell_2}$, which is exponential for bounded arity, and doubly exponential for unbounded arity of predicates in $P$.

Each rule from $\Gamma$ can participate in a hyperresolution inference with ground clauses in $k^b$ ways, which is exponential in $|P|$. Furthermore, by Theorem 7.4.2 the number of rules $t$ in $\Gamma$ is exponential in $|KB| + |P|$. Hence, the number of hyperresolution inference steps is bounded by $tk^b = t \cdot 2^{b\cdot(\ell_1+\ell_2)}$. For bounded arity of predicates in $P$, this number is exponential, and doubly exponential otherwise. Hence, in the same way as in Lemma 6.2.5, the number of concrete domain inference steps is exponential for bounded arity of predicates in $P$, and doubly exponential otherwise, thus implying the claim of the theorem. □

Note that most applications require predicates of small arity. Hence, the assumption that the arity of predicates is bounded is realistic in practice, thus giving a worst-case optimal algorithm.

## 9.6 Related Work

$\mathcal{AL}$-log [39] is a hybrid logic that combines an $\mathcal{ALC}$ knowledge base with datalog rules, where the latter can be constrained with unary atoms having $\mathcal{ALC}$ concepts as predicates in rule bodies. Query answering in $\mathcal{AL}$-log is decided by a variant of constrained resolution combined with a tableau algorithm for $\mathcal{ALC}$. The combined algorithm is shown to run in single nondeterministic exponential time. Because atoms with concept predicates can occur only as constraints in rule bodies, the rules are

applicable only to explicitly named objects. Our restriction to DL-safe rules has the same effect. However, our approach is more general in the following ways: (*i*) it supports a more expressive description logic, (*ii*) it allows using both concepts and roles in DL-atoms, and (*iii*) DL-atoms can be used in rule heads as well. Furthermore, we present a query answering algorithm as an extension of deductive database techniques running in deterministic exponential time.

A comprehensive study of combining datalog rules with description logics was presented in [84]. The logic considered is $\mathcal{ALCNR}$, which, although less expressive than $\mathcal{SHIQ}$, provides constructors characteristic of most DL languages. The results of the study can be summarized as follows: (*i*) answering conjunctive queries over $\mathcal{ALCNR}$ knowledge bases is decidable, (*ii*) query answering in the extension of $\mathcal{ALCNR}$ with nonrecursive datalog rules, where both concepts and roles can occur in rule bodies, is also decidable, because it can be reduced to computing a union of conjunctive query answers, (*iii*) if rules are recursive, query answering becomes undecidable, (*iv*) decidability can be regained by disallowing certain combinations of constructors in the logic, and (*v*) decidability can be regained by requiring rules to be *role-safe*, where at least one variable from each role literal must occur in a non-DL-atom. As in $\mathcal{AL}$-log, query answering is decided using constrained resolution and a modified version of the tableau calculus. Apart from treating a more expressive logic, in our approach all variables in a rule must occur in at least one non-DL-atom, but concepts and roles are allowed to occur in rule heads. Hence, when compared to the variant (*v*), our approach is slightly less general in some, and slightly more general in other aspects.

The Semantic Web Rule Language (SWRL) [69] combines OWL-DL with rules in which concept and role predicates are allowed to occur in the head and in the body without any restrictions. Hence, apart from technicalities such as allowing concept expressions to occur in rules, SWRL is compatible with DL-rules. As mentioned before, this combination is undecidable but, as pointed out by the authors, (incomplete) reasoning in such a logic can be performed using general first-order theorem provers. By restricting DL-safe rules to contain only DL-atoms and the special predicate $\mathcal{O}$, we obtain a proper subset of SWRL, in which expressivity is traded for decidability. Also, we provide an optimal query answering algorithm for a fragment of SWRL.

An approach for combining rules and description logics under a nonmonotonic semantics was presented in [123]. To achieve decidability, the author also employs DL-safety. Furthermore, rules are allowed to contain non-DL-atoms under negation-as-failure, which is interpreted under stable-model semantics. Finally, unique name assumption was assumed in [123], but this was later dropped in [124]. Hence, for programs without negation-as-failure and for answering positive ground queries, this approach coincides with ours. Also, in [123], a reasoning algorithm was presented, which don't-know nondeterministically reduces satisfiability and query answering w.r.t. combined knowledge bases to satisfiability of a DL knowledge base. Due to a huge amount guessing, such an algorithm is likely to be unsuitable for practice; on the contrary, we present a deterministic reasoning algorithm that can be combined with existing optimization techniques of disjunctive deductive databases.

Another approach for combining answer set programming with description logics was presented in [44]. The interaction between the subsystems is enabled by exchanging only unit ground consequences between the two components. The set of derivable facts is obtained by fixpoint computation. In this approach, the two systems are not tightly integrated, since the interaction between the systems is performed only through the exchange of unit consequences, resulting in a semantics that is not compatible with the first-order semantics. In the example in Section 9.4, this approach cannot derive that *Oedipus* is a child from the fact that *Oedipus* is a *GoodChild* or a *BadChild*$'$.

The approaches from [57] and [148] for reducing certain DL fragments to logic programming can easily be extended with rules, by simply appending the rules to the result of the transformation. However, the DL considered there does not support existential quantifiers, negation, or disjunction under positive polarity, so it is significantly less expressive than Horn-$\mathcal{SHIQ}(\mathbf{D})$. Hence, our approach is a proper extension.

# Chapter 10

# Answering Conjunctive Queries

Conjunctive queries were introduced in [32] as a formalism capable of expressing the class of selection–projection–join–renaming relational queries [1]. The vast majority of relational queries used in practice falls into this fragment, so a great deal of database research has been devoted to devising efficient algorithms for query answering and deciding query containment.

Because conjunctive queries have been found useful in diverse practical applications, it is natural to use them as an expressive formalism for querying description logic knowledge bases. Algorithms for answering conjunctive queries over DL knowledge bases expressed in various DL variants were presented in [143, 28, 72]. A common approach used in these algorithms is to reduce the problem of query answering to standard DL reasoning problems, as this enables reusing existing DL reasoning systems and services.

It is well known that complex encodings usually yield relatively poor performance in practice. Hence, in this chapter, we extend the algorithms from Chapters 5 and 6 to obtain an algorithm for answering conjunctive queries that works directly with the query and the knowledge base, without any encoding. In this way, we obtain an algorithm suitable for practical application. Furthermore, to the best of our knowledge, this is the first attempt to realize conjunctive query answering over description logic knowledge bases in the framework of resolution.

## 10.1 Definition of Conjunctive Queries

We first define conjunctive queries over $\mathcal{ALCHIQ}(\mathbf{D})$ knowledge bases.

**Definition 10.1.1.** *Let KB be an $\mathcal{ALCHIQ}(\mathbf{D})$ knowledge base, and let $x_1, \ldots, x_n$ and $y_1, \ldots, y_m$ be sets of* distinguished *and* nondistinguished *variables, denoted with $\mathbf{x}$ and $\mathbf{y}$, respectively. A* conjunctive query *over KB, denoted with $Q(\mathbf{x}, \mathbf{y})$, is a finite conjunction of DL-atoms of the form $(\neg)A(s)$ or $R(s,t)$, for $A$ an atomic concept, $R$ an abstract role, and $s$ and $t$ individuals from KB or variables from $\mathbf{x}$ or $\mathbf{y}$. The inference problems for conjunctive queries are defined as follows:*

- Query answering: *An* answer *of a query* $Q(\mathbf{x}, \mathbf{y})$ *w.r.t. KB is an assignment* $\theta$ *of individuals to distinguished variables such that* $\pi(KB) \models_{\mathbf{D}} \exists \mathbf{y} : Q(\mathbf{x}\theta, \mathbf{y})$.
- Query containment: *A query* $Q_2(\mathbf{x}, \mathbf{y_2})$ *is contained in a query* $Q_1(\mathbf{x}, \mathbf{y_1})$ *w.r.t. KB if* $\pi(KB) \models_{\mathbf{D}} \forall \mathbf{x} : [\exists \mathbf{y_2} : Q_2(\mathbf{x}, \mathbf{y_2}) \rightarrow \exists \mathbf{y_1} : Q_1(\mathbf{x}, \mathbf{y_1})]$.

A few remarks regarding Definition 10.1.1 are in order.

**Negative Atoms.** Negative concept atoms are usually not allowed to occur in conjunctive queries. However, including such atoms in Definition 10.1.1 makes the following presentation simpler, and does not change the formalism in any significant way.

**Abstract Roles.** The restriction to abstract roles in queries has been introduced in order not to require c-factoring of the query, which introduces problems for our algorithms. Note that concrete roles can still be used in the knowledge base.

**Transitive Roles.** It is straightforward to extend Definition 10.1.1 to $\mathcal{SHIQ}(\mathbf{D})$ knowledge bases; however, handling transitivity in queries seems to be quite difficult in the framework developed in previous chapters. Namely, our algorithms deal with transitivity axioms by encoding a $\mathcal{SHIQ}(\mathbf{D})$ knowledge base $KB$ into a $\mathbf{D}$-equisatisfiable $\mathcal{ALCHIQ}(\mathbf{D})$ knowledge base $\Omega(KB)$. As mentioned in Section 5.2, this transformation does not preserve entailment of ground complex role atoms. Therefore, we prohibit the usage of complex roles in conjunctive queries (note that such roles can still be used in $KB$). Under this assumption, $\pi(KB) \models_{\mathbf{D}} \exists \mathbf{y} : Q(\mathbf{x}\theta, \mathbf{y})$ if and only if $\pi(\Omega(KB)) \models_{\mathbf{D}} \exists \mathbf{y} : Q(\mathbf{x}\theta, \mathbf{y})$, which allows us to assume in the rest of this chapter that $KB$ is an $\mathcal{ALCHIQ}(\mathbf{D})$ knowledge base.

**(In)equality DL-Atoms.** Contrary to Definition 9.2.1 that defines DL-atoms for DL-rules, Definition 10.1.1 does not allow the (in)equality predicate to occur in DL-atoms of conjunctive queries. Namely, positive equality literals do not increase the expressivity of conjunctive queries: for a first-order formula $\varphi$, a variable $x$, and a term $t$ that does not contain $x$ as a proper subterm, the formulae $\exists x : (\varphi \wedge x \approx t)$ and $\varphi\{x \mapsto t\}$ are equisatisfiable [99]. Furthermore, based on unpublished results from [30], allowing negative equality literals to occur in a conjunctive query in all likelihood makes query answering undecidable.

**Conjunctive Queries vs. DL-safe Rules.** Querying DL knowledge bases might be performed using DL-safe rules, presented in Chapter 9; however, DL-safe rules are less expressive than conjunctive queries. To understand the practical implications of using either formalism, consider $KB = \{\exists hasParent.Person(Peter)\}$, and a query retrieving "all objects having a parent." Note that $KB$ states that *Peter* has a parent, but does not reveal his identity (that is, the name of the parent is not in the ABox).

Our question corresponds to the conjunctive query $Q(x, y) = \exists y : hasParent(x, y)$. Obviously, $KB \models \exists y : hasParent(Peter, y)$, so *Peter* is an answer to $Q(x, y)$ in $KB$.

Our question also corresponds to the DL-safe rule $Q(x) \leftarrow hasParent(x, y), \mathcal{O}(y)$; the literal $\mathcal{O}(y)$ is required in the rule body to ensure DL-safety. For $P$ a program containing the rule, $(KB, P) \not\models Q(Peter)$: namely, the name of the father is not known, and is therefore not contained in the predicate $\mathcal{O}$. The requirement on DL-safety requires $y$ to be bound to a named individual, which is the same as making $y$ a distinguished variable in the conjunctive query. Hence, querying by DL-safe rules is the same as using conjunctive queries without nondistinguished variables.

In certain circumstances, it is possible to use DL-safe rules for querying without having to completely give up on nondistinguished variables. In [75], the authors present the *query roll-up* technique, by means of which certain tree-like queries with nondistinguished variables can be transformed to queries without nondistinguished variables. In the previous example, this amounts to adding an axiom $\exists R.C \sqsubseteq D$ to $KB$, and then answering the DL-safe query $Q(x) \leftarrow D(x), \mathcal{O}(x)$. As discussed in [75], query roll-up is easy for conjunctive queries where nondistinguished variables do not occur in cycles of the query graph (as defined in the following section). Hence, conjunctive queries are more expressive than DL-safe rules only if they contain cycles involving nondistinguished variables; furthermore, dealing with such cycles is the main issue that the algorithm presented in this chapter has to deal with.

## 10.2 Answering Conjunctive Queries

Let $KB$ be an $\mathcal{ALCHIQ}(\mathbf{D})$ knowledge base. For a conjunctive query $Q(\mathbf{x}, \mathbf{y})$, the assignment $\theta$ such that $\theta\mathbf{x} = \mathbf{a}$ is an answer to the query w.r.t. $KB$ if and only if the set of closures $\Gamma' = \Xi(KB) \cup \{\neg Q(\mathbf{a}, \mathbf{y})\}$ is unsatisfiable, where $\neg Q(\mathbf{a}, \mathbf{y})$ is a closure obtained by negating each conjunct of $Q(\mathbf{a}, \mathbf{y})$. In the following, we show how to decide satisfiability of $\Gamma'$ by basic superposition. For that purpose, we first define the notion of a query graph:

**Definition 10.2.1.** *A* query graph *for a conjunctive query $Q(\mathbf{a}, \mathbf{y})$ is a directed graph with the following structure:*

- *Each variable $y \in \mathbf{y}$ is associated with a unique node;*
- *Each occurrence of a constant in $Q(\mathbf{a}, \mathbf{y})$ is associated with a unique node (that is, different occurrences of the same constant are associated with distinct nodes);*
- *For each literal $(\neg)A(s) \in Q(\mathbf{a}, \mathbf{y})$, the node $s$ is labeled with $(\neg)A$;*
- *For each literal $R(s, t) \in Q(\mathbf{a}, \mathbf{y})$, the graph contains a directed arc labeled with $R$, pointing from $s$ to $t$.*

*Each conjunctive query defines a distinct query graph, so we do not make an explicit distinction between the two. Hence, by saying that a query is connected, tree-like, or acyclic, we refer to the properties of the query graph.*

Without loss of generality, we can assume that a query graph is weakly connected; that is, for each two nodes $s$ and $t$ in the graph, there is a path either from $s$ to $t$, or from $t$ to $s$. Namely, assume that a query $Q(\mathbf{a}, \mathbf{y})$ can be split into $n$ weakly connected, mutually disjoint subqueries $Q_1(\mathbf{a_1}, \mathbf{y_1}), \ldots, Q_n(\mathbf{a_n}, \mathbf{y_n})$. Obviously, $\pi(KB) \models_{\mathbf{D}} \bigwedge_{1 \leq i \leq n} \exists \mathbf{y_i} : Q_i(\mathbf{a_i}, \mathbf{y_i})$ if and only if $\pi(KB) \models_{\mathbf{D}} \exists \mathbf{y_i} : Q_i(\mathbf{a_i}, \mathbf{y_i})$, for all $1 \leq i \leq n$. Splitting $Q(\mathbf{a}, \mathbf{y})$ into $Q_i(\mathbf{a_i}, \mathbf{y_i})$ can be performed in time that is polynomial in $|Q(\mathbf{a}, \mathbf{y})|$, so this assumption does not increase the complexity of reasoning.

A slight problem arises if $\neg Q(\mathbf{a}, \mathbf{y})$ contains constants that do not occur at substitution positions. Consider the following closures:

$$a_1 \approx a_1' \vee a_2 \approx a_2' \tag{10.1}$$

$$\neg Q_1(\mathbf{a_1}, \mathbf{y_1}) \tag{10.2}$$

$$\neg Q_2(\mathbf{a_2}, \mathbf{y_2}) \tag{10.3}$$

Assuming that $a_i \in \mathbf{a_i}$ and $a_i' \in \mathbf{a_i'}$ for $i \in \{1, 2\}$, one can perform superposition from (10.1) into (10.2), and then into (10.3), which produces the following closure:

$$\neg Q_1(\mathbf{a_1'}, \mathbf{y_1}) \vee \neg Q_2(\mathbf{a_2'}, \mathbf{y_2}) \tag{10.4}$$

Now (10.4) contains more variables than either of the premises it was derived from, and therefore causes termination problems.

We deal with this problem by applying the structural transformation to $\neg Q(\mathbf{a}, \mathbf{y})$. In particular, we replace $\Gamma'$ with $\Gamma$ defined as follows, where, for each $a \in \mathbf{a}$, $\mathcal{O}_a$ is a new predicate unique for $a$, $x_a$ is a new variable unique for $a$, and $\mathbf{x_a}$ is the vector of variables obtained from $\mathbf{a}$ by replacing each $a$ with $x_a$:

$$\Gamma = \Xi(KB) \cup \{\neg Q(\mathbf{x_a}, \mathbf{y}) \vee \bigvee_{a \in \mathbf{a}} \neg \mathcal{O}_a(x_a)\} \cup \bigcup_{a \in \mathbf{a}} \{\mathcal{O}_a(a)\}$$

The sets of closures $\Gamma'$ and $\Gamma$ are obviously equisatisfiable. In the rest of this section, we write $\neg \mathcal{O}_{\mathbf{a}}(\mathbf{x_a})$ for $\bigvee_{a \in \mathbf{a}} \neg \mathcal{O}_a(x_a)$.

This transformation solves the problem in the following way. By Definition 10.2.2, the literals $\mathcal{O}_a(x_a)$ are selected, so a closure (10.5) can participate only in resolution with closures of the form (10.6) to produce a closure of the form (10.7):

$$\neg Q(\mathbf{x_a}, \mathbf{y}) \vee \bigvee_{a \in \mathbf{a}} \neg \mathcal{O}_a(x_a) \tag{10.5}$$

$$\mathcal{O}_a(a) \tag{10.6}$$

$$\neg Q([\mathbf{a}], \mathbf{y}) \tag{10.7}$$

In (10.7), all constants $[\mathbf{a}]$ are marked, which prevents superposition inferences from (10.1) into (10.7).

We now define the calculus for checking satisfiability of $\Gamma$:

**Definition 10.2.2.** *With $\mathcal{BS}^{\mathbf{D},+}_{CQ}$ we denote the $\mathcal{BS}^{\mathbf{D}}$ calculus, extended with decomposition, and parameterized as follows:*

- *Inference conclusions, whenever possible, are decomposed according to the following table, where $s$ is a term of the form $f_1(\dots f_m(u)\dots)$ with $u$ a variable or a constant, and $t_i$ are terms of the form $f_{i,1}(\dots f_{i,m}(x)\dots)$ for $m \geq 1$:*

$$\begin{array}{rcl} D \cdot \rho \vee R([s], [f(s)]) & \rightsquigarrow & \begin{array}{c} D \cdot \rho \ \vee \ Q_{R,f}([s]) \\ \neg Q_{R,f}(x) \ \vee \ R(x, [f(x)]) \end{array} \\ D \cdot \rho \vee R([f(s)], [s]) & \rightsquigarrow & \begin{array}{c} D \cdot \rho \ \vee \ Q_{\mathsf{Inv}(R),f}([s]) \\ \neg Q_{\mathsf{Inv}(R),f}(x) \ \vee \ R([f(x)], x) \end{array} \\ (\neg)A_1([t_1]) \vee \ldots \vee (\neg)A_n([t_n]) & \rightsquigarrow & \begin{array}{c} Q_{(\neg)A_1,t_1}(x) \vee \ldots \vee Q_{(\neg)A_n,t_n}(x) \\ \neg Q_{(\neg)A_i,t_i}(x) \vee (\neg)A_i([t_i]),\ 1 \leq i \leq n \end{array} \end{array}$$

- *The precedence for LPO is $f > c > P > Q_{R,f} > \mathsf{T}$ for each function symbol $f$, constant $c$, nondefinition predicate $P$, and definition predicate $Q_{R,f}$.*
- *If a closure $C$ contains a literal $\neg\mathcal{O}_a(x_a)$, then all such literals are selected; otherwise, all negative binary literals are selected.*

We extend the $\mathcal{ALCHIQ}(\mathbf{D})$-closures to closures obtained in a saturation of $\Gamma$ by $\mathcal{BS}^{\mathbf{D},+}_{CQ}$:

**Definition 10.2.3.** *The class of $\mathcal{CQ}$-closures w.r.t. a conjunctive query $Q(\mathbf{a}, \mathbf{y})$ over an $\mathcal{ALCHIQ}(\mathbf{D})$ knowledge base KB is obtained as a generalization of closures from Tables 5.2 and 6.2 with the following changes:*

- *Conditions (*iii*)–(*vi*) are dropped.*
- *Closure types 5 and 6 are replaced with a new type $5'$, which contains each closure $C$ satisfying all of the following conditions:*
  1. *$C$ contains only equality or unary literals;*
  2. *$C$ contains only one variable $x$;*
  3. *The depth of a term in $C$ is bounded by the number of literals in $Q(\mathbf{a}, \mathbf{y})$;*
  4. *If $C$ contains a term of the form $f(t)$, then all terms of the same depth in $C$ are of the form $g(t)$, and all terms of smaller depth are (not necessarily proper) subterms of $t$;*
  5. *Only the outmost position of a term in $C$ can be unmarked; that is, a term containing a function symbol is either of the form $[f(t)]$ or of the form $f([t])$;*
  6. *Equality and inequality literals in $C$ can have the form $[f(t)] \circ [g(t)]$ or $[f(g(t))] \circ [t]$ for $\circ \in \{\approx, \not\approx\}$.*

- *Closure type 8 is modified to allow unary and (in)equality literals to contain unary terms whose depth is bounded by the number of literals in $Q(\mathbf{a}, \mathbf{y})$; only outermost positions in a term can be unmarked; and all (in)equality literals are of the form $[f(a)] \circ [b]$, $[f(t)] \circ [g(t)]$, $[f(g(t))] \circ [t]$ or $\langle a \rangle \circ \langle b \rangle$, for $\circ \in \{\approx, \not\approx\}$ and $t$ a ground term.*

- *A new* query closure *type contains closures of the form $\neg Q([\mathbf{a}], \mathbf{y}) \vee C_8$, where $Q([\mathbf{a}], \mathbf{y})$ is weakly connected, it contains at least one binary literal, and $C_8$ is a possibly empty closure of type 8, which is empty if $\mathbf{a}$ is empty.*

- *A new* initial closure *type contains closures of the form $\neg \mathcal{O}_{\mathbf{a}}(\mathbf{x}_{\mathbf{a}}) \vee \neg Q(\mathbf{x}_{\mathbf{a}}, \mathbf{y})$.*

We now show that saturation of $\Gamma$ by $\mathcal{BS}^{\mathbf{D},+}_{CQ}$ terminates for each $\Gamma$, thus yielding a decision procedure for conjunctive query answering:

**Theorem 10.2.4.** *For a conjunctive query $Q(\mathbf{a}, \mathbf{y})$ over an $\mathcal{ALCHIQ}(\mathbf{D})$ knowledge base $KB$, saturation of $\Gamma$ by $\mathcal{BS}^{\mathbf{D},+}_{CQ}$ decides satisfiability of $\Gamma$ in time doubly exponential in $|KB| + |Q(\mathbf{a}, \mathbf{y})|$, assuming a bound on the arity of the concrete domain predicates, and for unary coding of numbers in input.*

*Proof.* As in Lemma 5.3.5, Condition 5 of Definition 10.2.3 ensures that, in each $\mathcal{CQ}$-closure of type $5'$, the maximal literal contains the deepest term of a closure. This allows us to show the following property (*): application of a $\mathcal{BS}^{\mathbf{D},+}_{CQ}$ inference to $\mathcal{CQ}$-closures results in one or more $\mathcal{CQ}$-closures.

Namely, hyperresolution with a closure of type 7 is possible only with closures of types 3, 4, and 8, and results in a closure of type $5'$ or 8. Furthermore, each conclusion of a superposition inference into a generator closure is decomposed into a generator and a closure of type $5'$; eligibility of the inference for decomposition follows by the same argument as in Theorem 5.4.8. Note that, since $R([f(t)], [t])$ and $\mathsf{Inv}(R)([t], [f(t)])$ are logically equivalent, the predicate $Q_{\mathsf{Inv}(R),f}$ can be used as the definition predicate for $R([f(x)], x)$. Finally, since only the outermost position of any term can be unmarked, two terms $t_1$ and $t_2$ can be unified only at the outer positions. Since both terms are unary, they can be unified only if one of them is of the form $f_1(\ldots f_i(x)\ldots)$, and the other one is of the form $f_1(\ldots f_i(\ldots f_n(x')\ldots)\ldots)$, so the unifier is of the form $x \mapsto f_{i+1}(\ldots f_n(x')\ldots)$. Therefore, the maximal term depth in the conclusion is $n$, and the conclusion is a $\mathcal{CQ}$-closure.

Consider an inference with an initial closure of the form $\neg \mathcal{O}_{\mathbf{a}}(\mathbf{x}_{\mathbf{a}}) \vee \neg Q(\mathbf{x}_{\mathbf{a}}, \mathbf{y})$. For each variable $x_a$, such a closure contains a literal $\neg \mathcal{O}_a(x_a)$, all of which are selected. Hence, the only possible inference is hyperresolution on all $\neg \mathcal{O}_a(x_a)$ with ground premises of the form $\mathcal{O}_a(a) \vee C$, which yields a query closure of the form $\neg Q([\mathbf{a}], \mathbf{y}) \vee C_8$.

A more complex case is an inference with a query closure $\neg Q([\mathbf{a}], \mathbf{y}) \vee C_8$. All constants in such a closure are marked, so superposition into it is not possible. Furthermore, since the closure always contains at least one binary literal, it can only participate as the main premise in hyperresolution on all binary literals, with a unifier

$\sigma$, and with side premises $E_i$ being of type 3, 4, or 8. For side premises of types 3 and 4, with $x_i$ we denote the free variable of the $i$-th side premise.

Assume that at least one side premise is of type 8, or that $Q([\mathbf{a}], \mathbf{y})$ contains a constant term. Then, some term in $Q([\mathbf{a}], \mathbf{y})\sigma$ is a ground term $\alpha$. Let $E_i$ be the side premise matched to the binary literal containing $\alpha$ and some other term $\beta$. If $E_i$ is ground, then $\beta\sigma$ is obviously a ground term. On the contrary, if $E_i$ is nonground, since the maximal literal of $E_i$ is of the form $R(x_i, \langle f(x_i)\rangle)$ or $R([f(x_i)], x_i)$, $E_i\sigma$ is ground, so $\beta\sigma$ is again a ground term. Since $Q([\mathbf{a}], \mathbf{y})$ is weakly connected, constants are propagated to the entire query, so $Q([\mathbf{a}], \mathbf{y})\sigma$ is ground. Since all terms containing function symbols in side premises are of the form $f_i(x_i)$, and they are of depth one, the maximal depth of a term after unification is bounded by the maximal length of a path in $Q([\mathbf{a}], \mathbf{y})$, which is bounded by the number of binary literals in $Q([\mathbf{a}], \mathbf{y})$. Hence, the conclusion is a $\mathcal{CQ}$-closure of type 8.

Assume that no side premise is of type 8, and that $Q([\mathbf{a}], \mathbf{y})$ does not contain a constant; then, $C_8$ is empty, and we write simply $Q(\mathbf{y})$. Since $Q(\mathbf{y})$ is weakly connected, similarly as in the previous case, we conclude that the unifier $\sigma$ contains mappings of the form $x_i \mapsto s_i$ and $y_i \mapsto t_i$, where $s_i$ and $t_i$ are terms of the form $f_{i,1}(\ldots f_{i,m}(x)\ldots)$, and $m$ is bounded by the maximal length of a path in $Q(\mathbf{y})$. Hence, the hyperresolution conclusion $C$ contains only unary literals of the form $(\neg)A([t_i])$, where $t_i$ is of the form $f_{i,1}(\ldots f_{i,m}(x)\ldots)$ and $m$ is bounded by the number of binary literals in $Q(\mathbf{y})$. Since the conclusion does not have equality literals, it satisfies Conditions 1, 2, 3, 5, and 6 of the $\mathcal{CQ}$-closure type 5′; however, terms $t_i$ need not satisfy Condition 4. However, the closure is decomposed by $\mathcal{BS}^{\mathbf{D},+}_{CQ}$ into several $\mathcal{CQ}$-closures. Since in the LPO we ensure that $A > Q_{(\neg)A_i,t_i}$, we also have that $\neg Q_{(\neg)A_i,t_i}(x) \prec (\neg)A([t_i])$; furthermore, $(\neg)A([t_i])$ originates from some side premise $E_j$, so the inference is eligible for decomposition by Proposition 5.4.6.

This covers all $\mathcal{BS}^{\mathbf{D},+}_{CQ}$ inferences on $\mathcal{CQ}$-closures, so the property (*) holds.

Let $p$ and $f$ be the number of predicates and function symbols occurring in $\Xi(KB)$; both are linear in $|KB|$ for unary coding of numbers. The number $p'$ of predicates $Q_{S,f}$, introduced by decomposition after superposition into a generator, is quadratic in $|KB|$. For $n$ the number of literals in $Q(\mathbf{a}, \mathbf{y})$, the number of terms of the form $f_1(\ldots f_i(x)\ldots)$ is bounded by $\ell = f + f^2 + \ldots + f^n$, which is exponential in $|KB| + |Q(\mathbf{a}, \mathbf{y})|$.

Consider a hyperresolution inference with a query closure. The conclusion of such an inference is decomposed only if the conclusion is nonground, which is possible only if all premises are of type 3 or 4. Therefore, the hyperresolution conclusion can only contain predicates from $\Xi(KB)$ and definition predicates $Q_{S,f}$. The number of such predicates is bounded by $2(p + p')$ (the factor 2 takes into account that unary literals can occur positively or negatively), so the number of predicates $Q_{(\neg)A,t}$ introduced by decomposition after resolution with a query closure is bounded by $\wp = 2(p+p')\ell$, which is exponential in $|KB| + |Q(\mathbf{a}, \mathbf{y})|$. Furthermore, the number of literals in the longest closure of type 5′ is bounded by $2(\wp+p+p')\ell$, which is exponential in $|KB|+|Q(\mathbf{a}, \mathbf{y})|$. Similarly, the maximal number of ground terms containing function symbols is $c \cdot \ell$,

where $c$ is the number of constants in $\Xi(KB)$, which gives an exponential bound on the length of the maximal closure of type 8. In all cases, a maximal closure is at most exponential in length, so the number of possible $\mathcal{CQ}$-closures is doubly exponential in $|KB| + |Q(\mathbf{a}, \mathbf{y})|$.

For $\Gamma$ as assumed by the theorem, all closures in $\Gamma$ are $\mathcal{CQ}$-closures: closures from $\Xi(KB)$ are $\mathcal{ALCHIQ}$-closures, and $\neg\mathcal{O}_{\mathbf{a}}(\mathbf{x_a}) \vee \neg Q(\mathbf{x_a}, \mathbf{y})$ is either an initial or a query closure (depending on whether $\mathbf{x_a}$ is empty), or it is of type $5'$, 7, or 8. By property (*), in any derivation $\Gamma = N_0, \ldots, N_i$, each set $N_i$ contains only $\mathcal{CQ}$-closures. Since the number of closures in each $N_i$ is at most doubly exponential in $|KB| + |Q(\mathbf{a}, \mathbf{y})|$, saturation by $\mathcal{BS}_{CQ}^{\mathbf{D},+}$ terminates in doubly exponential time. Since all decomposition inferences are eligible for decomposition, by Theorem 5.4.4 $\mathcal{BS}_{CQ}^{\mathbf{D},+}$ is sound and complete, so it decides satisfiability of $\Gamma$. □

Note that, by assuming a bound on $|Q(\mathbf{x}, \mathbf{y})|$, the query answering algorithm becomes exponential. Namely, the depth of the terms of the form $f_1(\ldots f_i(x)\ldots)$ is then polynomial in $|KB|$, so $\ell$ from the proof of Theorem 10.2.4 becomes polynomial.

## 10.3 Deciding Conjunctive Query Containment

We now apply the query answering algorithm to decide query containment.

**Theorem 10.3.1.** *Let $Q_1(\mathbf{x}, \mathbf{y_1})$ and $Q_2(\mathbf{x}, \mathbf{y_2})$ be two conjunctive queries with the same set of distinguished variables, defined over an $\mathcal{ALCHIQ}(\mathbf{D})$ knowledge base $KB$. The query $Q_2(\mathbf{x}, \mathbf{y_1})$ is contained in the query $Q_1(\mathbf{x}, \mathbf{y_2})$ w.r.t. $KB$ if $\mathbf{a}$ is an answer to $Q_1(\mathbf{x}, \mathbf{y_1})$ over $KB \cup \{Q_2(\mathbf{a}, \mathbf{b})\}$, where $\mathbf{a}$ and $\mathbf{b}$ are sets of new distinct individuals, not occurring in $Q_1(\mathbf{x}, \mathbf{y_1})$, $Q_2(\mathbf{x}, \mathbf{y_2})$, and $KB$.*

*Proof.* The claim can easily be established by transforming the definition of query containment using the following well-known equivalences:

$$\begin{array}{rl}
\pi(KB) \models_{\mathbf{D}} \forall \mathbf{x} : [\exists \mathbf{y_2} : Q_2(\mathbf{x}, \mathbf{y_2}) \rightarrow \exists \mathbf{y_1} : Q_1(\mathbf{x}, \mathbf{y_1})] & \Leftrightarrow \\
\pi(KB) \cup \{\neg\forall \mathbf{x} : [\neg\exists \mathbf{y_2} : Q_2(\mathbf{x}, \mathbf{y_2}) \vee \exists \mathbf{y_1} : Q_1(\mathbf{x}, \mathbf{y_1})]\} \text{ is } \mathbf{D}\text{-unsatisfiable} & \Leftrightarrow \\
\pi(KB) \cup \{\exists \mathbf{x}, \mathbf{y_2} : Q_2(\mathbf{x}, \mathbf{y_2}) \wedge \forall \mathbf{y_1} : \neg Q_1(\mathbf{x}, \mathbf{y_1})\} \text{ is } \mathbf{D}\text{-unsatisfiable} & \Leftrightarrow \\
\pi(KB) \cup \{Q_2(\mathbf{a}, \mathbf{b}), \forall \mathbf{y_1} : \neg Q_1(\mathbf{a}, \mathbf{y_1})\} \text{ is } \mathbf{D}\text{-unsatisfiable} & \Leftrightarrow \\
\pi(KB) \cup \{Q_2(\mathbf{a}, \mathbf{b})\} \models_{\mathbf{D}} \exists \mathbf{y_1} : Q_1(\mathbf{a}, \mathbf{y_1})\} & \Leftrightarrow \\
\mathbf{a} \text{ is an answer to } Q_1(\mathbf{x}, \mathbf{y_1}) \text{ over } KB \cup \{Q_2(\mathbf{a}, \mathbf{b})\} &
\end{array}$$

The constants $\mathbf{a}$ and $\mathbf{b}$ are introduced by skolemizing $\exists \mathbf{x}, \mathbf{y_2}$. □

The following corollary follows immediately from Theorem 10.3.1:

**Corollary 10.3.2.** *Let $Q_1(\mathbf{x}, \mathbf{y_1})$ and $Q_2(\mathbf{x}, \mathbf{y_2})$ be conjunctive queries defined over an $\mathcal{ALCHIQ}(\mathbf{D})$ knowledge base $KB$. Then, deciding if $Q_2(\mathbf{x}, \mathbf{y_2})$ is contained in a query $Q_1(\mathbf{x}, \mathbf{y_1})$ w.r.t. $KB$ by saturation with $\mathcal{BS}_{CQ}^{\mathbf{D},+}$ runs in time that is doubly exponential in $|KB| + |Q_1(\mathbf{x}, \mathbf{y_1})| + |Q_2(\mathbf{x}, \mathbf{y_2})|$, assuming a bound on the arity of the concrete domain predicates and for unary coding of numbers in input.*

## 10.4 Related Work

Conjunctive queries were introduced in [32] as a query language for the relational model, capable of expressing a large number of practically relevant queries. The problem of deciding conjunctive query containment was shown to be NP-complete, using an algorithm based on deciding whether a homomorphism—an embedding of nondistinguished variables—between the subsumed and the subsuming query exists. Furthermore, the authors identify the close relationship between query answering and query containment. In practice, query equivalence is used extensively for query optimization in relational databases: a complex query is usually transformed into an equivalent but simpler query, which can be executed more efficiently [1].

In [135], conjunctive queries were extended to *recursive queries*, for which deciding equivalence was shown to be undecidable. However, note that this is so only if the minimal fixpoint semantics is assumed for the queries. Such a semantics is common in logic programming; however, it is principally different from the first-order semantics. Hence, the results from [135] do not apply in our case.

Answering conjunctive queries and conjunctive query containment over description logic knowledge bases was studied in the CARIN system [84]. The description logic considered is $\mathcal{ALCNR}$, and is significantly less expressive than $\mathcal{SHIQ}(\mathbf{D})$. The decision procedure for query answering is based on constrained resolution, which combines SLD-resolution backward chaining with tableau reasoning.

In [28], the authors study the containment of conjunctive queries over constraints, expressed in the description logic $\mathcal{DLR}_{reg}$. This logic is distinguished by allowing for $n$-ary relations and regular expressions over projections of relations. The technique used to handle query containment is based on a reduction to satisfiability of $\text{CPDL}_g$ (propositional dynamic logic with converse and graded modalities) programs. A similar technique was used to derive a procedure for rewriting queries over description logics using views in [29]. However, as pointed out in [50], the presented algorithm is incorrect in the presence of transitive roles.

In [72], a procedure for deciding containment of conjunctive queries over constraints, based on the reduction to $\mathcal{SHIQ}$, was presented. In this way the authors obtain a practical procedure. Namely, they argue that the approach from [28] is not practical, since a reasoner for $\text{CPDL}_g$ does not exists.

In [75], conjunctive queries have been proposed as the query language for the Semantic Web. The approach presented there is restricted only to tree-like queries, possibly containing constants. However, the approach was generalized later in [143] to an algorithm for answering conjunctive queries over $\mathcal{SH}f$ knowledge bases, and is thus the first algorithm for answering conjunctive queries over a logic with transitive roles. The algorithm is based on a reduction of query answering to standard reasoning problems in $\mathcal{SHIQ}$.

Contrary to the approach from [143], our approach supports inverse roles, but it allows only simple roles to occur in the queries. To the best of our knowledge, this is the first algorithm for query answering and query containment based on resolution.

# Chapter 11

# The Semantics of Metamodeling

A common practice in DL modeling is to divide a model into an *intensional* and an *extensional* part. The intensional part is analogous to a database schema, and it describes the general structure and the regularities of the world; the extensional part is analogous to a database instance, and it describes a particular state of the world.

To better understand this duality, consider the following example, originally presented in [149]; a similar example can be found in [132]. A natural way to represent kinship between animal species is to organize them in a hierarchy of concepts. For example, the concept *Bird* represents the set of all birds, and the concept *Eagle* is a subconcept of *Bird*. The subconcept relationship states that all eagles are birds. This is an example of intensional knowledge, as it is concerned with defining the general notions of birds and eagles. Knowledge about concrete animals is extensional; for example, we may state that the individual *Harry* is an instance of the concept *Eagle*. Now the intensional knowledge implies that *Harry* is a *Bird* as well.

However, one might also make statements about individual species, such as "Eagles are listed in the IUCN Red List[1] of endangered species." Note an important distinction: we do not say that each individual eagle is listed in the Red List, but that the eagle species as a whole is. Hence, we may introduce a concept *RedListSpecies*, but this raises some concerns about the proper relationship between *RedListSpecies* and *Eagle*. Making the former a superconcept of the latter is incorrect, as it would imply that *Harry* is a *RedListSpecies*—clearly an undesirable conclusion. It is better to say that *Eagle* is *type of RedListSpecies*. Thus, *RedListSpecies* acts as a *metaconcept* for *Eagle*. Modeling with metaconcepts we call *metamodeling*.

Metamodeling can be used to build concise models if we axiomatize the properties of metaconcepts. For example, by stating that "It is not allowed to hunt the individuals of species listed in the Red List," we formalize the logical properties of the metaconcept *RedListSpecies*, which allows us to deduce that "It is not allowed to hunt *Harry*."

The examples, such as the one given previously, are often dismissed with an argument that "eagle as a species" and "eagle as a set of all individual eagles" are not the

[1] `http://www.redlist.org/`

one and the same thing, and should therefore not be referred to using the same symbol. Whereas an in-depth philosophical investigation might provide a more definitive answer, we believe that the notion of an "eagle" in most people's minds invokes a notion of a "mighty bird of prey." The interpretation of this notion as a concept or as an individual is of secondary concern, and is often context-dependent, so using different symbols for the same intuitive notion makes the model unnecessarily complex.

Metamodeling is supported in OWL-Full [106], the most expressive language of the OWL family. However, its semantics is controversial, mainly because it is nonstandard, which makes realizing practical reasoning systems difficult [68]. Therefore, OWL-DL was conceived as a "well-behaved" subset of OWL-Full by imposing the following restrictions: (*i*) logical and metalogical symbols are strictly separated, (*ii*) the symbols used as concepts, roles, and individuals are strictly separated, and (*iii*) restrictions required to yield a decidable logic, such as using only simple roles in number restrictions [73], are enforced. These restrictions make OWL-DL a syntactical variant of the $\mathcal{SHOIN}(\mathbf{D})$ description logic, which is decidable. This is desirable since, to practically implement reasoners for expressive logics, advanced optimization techniques are essential, and these are much easier to develop for decidable logics [4, Chapter 9].

Since it does not enforce (*iii*), OWL-Full is trivially undecidable. To obtain a decidable logic supporting metamodeling, it is natural to ask whether OWL-DL, extended with metamodeling in the style of OWL-Full, remains decidable. However, in Section 11.1 we show that even the basic description logic $\mathcal{ALC}$ becomes undecidable if restrictions (*i*) and (*ii*) are not enforced.

We analyze this result, and show that it is actually due to (*i*)—that is, to free mixing of logical and metalogical symbols. In a way, metamodeling in OWL-Full goes beyond its original purpose, and allows the user to tamper with the semantics of the modeling primitives themselves. In Section 11.2 we address this issue and present two alternative semantics: a *contextual* or $\pi$-semantics, which is essentially first-order, and a *HiLog* or $\nu$-semantics, which is based on HiLog [33]—a logic that provides a second-order flavor to first-order logic. We show that, under some technical assumptions, both semantics can be combined with $\mathcal{SHIQ}(\mathbf{D})$, yielding a decidable fragment of OWL-Full. We show that this does not increase the complexity of reasoning, so our results may provide a basis for practical implementation. Finally, in Section 11.3 we discuss the added expressivity of metamodeling on a concrete example.

## 11.1 Undecidability of Metamodeling in OWL-Full

In this section we show that the style of metamodeling adopted in OWL-Full leads to undecidability, even if combined with a very simple description logic $\mathcal{ALC}$. We start by presenting the definition of the $\mathcal{ALC}$-Full syntax and semantics. We base the semantics on the one of OWL-Full [106]; however, the latter is quite complex, so for readability we abstract from numerous technical details. We assume *rdf:*, *rdfs:* and *owl:* to be the RDF, RDFS, and OWL namespace prefixes, respectively [106].

Table 11.1: Semantics of $\mathcal{ALC}$-Full

| |
|---|
| 1. $\langle s, p, o\rangle \in KB$ implies $(s^I, o^I) \in \mathrm{EXT}^I(p^I)$ |
| 2. $\mathrm{CEXT}^I(\mathit{owl{:}Thing}^I) = \triangle^I$ |
| 3. $\mathrm{CEXT}^I(\mathit{owl{:}Nothing}^I) = \emptyset$ |
| 4. $(x, y) \in \mathrm{EXT}^I(\mathit{rdfs{:}subClassOf}^I)$ implies $\mathrm{CEXT}^I(x) \subseteq \mathrm{CEXT}^I(y)$ |
| 5. $(x, y) \in \mathrm{EXT}^I(\mathit{owl{:}sameAs}^I)$ implies $x = y$ |
| 6. $(x, y) \in \mathrm{EXT}^I(\mathit{owl{:}differentFrom}^I)$ implies $x \neq y$ |
| 7. $(x, y) \in \mathrm{EXT}^I(\mathit{owl{:}complementOf}^I)$ implies $\mathrm{CEXT}^I(x) = \triangle^I \setminus \mathrm{CEXT}^I(y)$ |
| 8. $(x, y) \in \mathrm{EXT}^I(\mathit{owl{:}unionOf}_1^I)$ and $(x, z) \in \mathrm{EXT}^I(\mathit{owl{:}unionOf}_2^I)$ imply $\mathrm{CEXT}^I(x) = \mathrm{CEXT}^I(y) \cup \mathrm{CEXT}^I(z)$ |
| 9. $(x, y) \in \mathrm{EXT}^I(\mathit{owl{:}intersectionOf}_1^I)$ and $(x, z) \in \mathrm{EXT}^I(\mathit{owl{:}intersectionOf}_2^I)$ imply $\mathrm{CEXT}^I(x) = \mathrm{CEXT}^I(y) \cap \mathrm{CEXT}^I(z)$ |
| 10. $(x, y) \in \mathrm{EXT}^I(\mathit{owl{:}someValuesFrom}^I)$ and $(x, p) \in \mathrm{EXT}^I(\mathit{owl{:}onProperty}^I)$ imply $\mathrm{CEXT}^I(x) = \{w \mid (w, z) \in \mathrm{EXT}^I(p) \wedge z \in \mathrm{CEXT}^I(y)\}$ |
| 11. $(x, y) \in \mathrm{EXT}^I(\mathit{owl{:}allValuesFrom}^I)$ and $(x, p) \in \mathrm{EXT}^I(\mathit{owl{:}onProperty}^I)$ imply $\mathrm{CEXT}^I(x) = \{w \mid (w, z) \in \mathrm{EXT}^I(p) \rightarrow z \in \mathrm{CEXT}^I(y)\}$ |

**Definition 11.1.1.** *Let $V$ be the* vocabulary *set consisting of these symbols:*

> *owl:Thing, owl:Nothing, rdf:type, rdfs:subClassOf, owl:sameAs, owl:differentFrom, owl:complementOf, owl:unionOf$_1$, owl:unionOf$_2$, owl:intersectionOf$_1$, owl:intersectionOf$_2$, owl:someValuesFrom, owl:allValuesFrom, owl:onProperty*

*Let $N$ be the set of* names *such that $V \subseteq N$. An $\mathcal{ALC}$-Full* knowledge base *$KB$ is a finite set of triples of the form $\langle s, p, o\rangle$, where $s, p, o \in N$.*

*An* interpretation *$I$ is a triple $(\triangle^I, \cdot^I, EXT^I)$, where $\triangle^I$ is a nonempty* domain set*, $\cdot^I : N \rightarrow \triangle^I$ is a* name interpretation function*, and $EXT^I : \triangle^I \rightarrow 2^{\triangle^I \times \triangle^I}$ is an* extension function*. Let $CEXT^I : \triangle^I \rightarrow 2^{\triangle^I}$ be the* concept extension function *defined as $CEXT^I(x) = \{y \mid (y, x) \in EXT^I(\mathit{rdf{:}type}^I)\}$. An interpretation $I$ is a* model *of $KB$ if it satisfies all conditions from Table 11.1. $KB$ is* satisfiable *if and only if a model of $KB$ exists.*

$\mathcal{ALC}$-Full differs from OWL-Full in that (*i*) it does not provide for concrete predicates; (*ii*) it does not include the *axiomatic triples* that define the semantics for built-in resources, such as *rdf:type*; and (*iii*) it allows only binary union and intersection. These distinctions are not relevant for our undecidability proof. In the rest of this section, we use $\langle a \sqcup b, p, o\rangle$ as a syntactic shortcut for the triples $\langle x, p, o\rangle$, $\langle x, \mathit{owl{:}unionOf}_1, a\rangle$, and $\langle x, \mathit{owl{:}unionOf}_2, b\rangle$, where $x$ is a new name. We use similar shortcuts for $\langle s, p, a \sqcup b\rangle$ and for $\sqcap$.

We show now that checking satisfiability of an $\mathcal{ALC}$-Full knowledge base $KB$ is undecidable by a reduction from the DOMINO TILING problem [20]. A *domino system* is a triple $\mathcal{D} = (D, H, V)$, where $D = \{D_1, \ldots, D_n\}$ is a finite set of *domino types*, $H \subseteq D \times D$ is the *horizontal compatibility relation*, and $V \subseteq D \times D$ is the *vertical compatibility relation*. A $\mathcal{D}$*-tiling* of an infinite grid is a function $t : \mathbb{N} \times \mathbb{N} \to D$ such that $t(0,0) = D_0$ and, for all $i, j \in \mathbb{N}$, $(t(i,j), t(i,j{+}1)) \in H$ and $(t(i,j), t(i{+}1,j)) \in V$. For a domino system $\mathcal{D}$, determining whether a $\mathcal{D}$-tiling exists is undecidable [20].

For a domino system $\mathcal{D}$, let $KB_\mathcal{D}$ be the $\mathcal{ALC}$-Full knowledge base containing axioms (11.1)–(11.9). Lemma 11.1.2 shows that $KB_\mathcal{D}$ exactly encodes the domino tiling problem.

$$\langle D_i \sqcap D_j, \mathit{rdfs{:}subClassOf}, \mathit{owl{:}Nothing}\rangle \text{ for } 1 \le i < j \le n \tag{11.1}$$

$$\langle \mathit{GRID}, \mathit{rdfs{:}subClassOf}, D_1 \sqcup \ldots \sqcup D_n\rangle \tag{11.2}$$

$$\langle \mathit{NotGRID}, \mathit{owl{:}complementOf}, \mathit{GRID}\rangle \tag{11.3}$$

$$\langle D_i, \mathit{rdfs{:}subClassOf}, \alpha_i\rangle, \langle \alpha_i, \mathit{owl{:}onProperty}, \mathit{owl{:}allValuesFrom}\rangle, \langle \alpha_i, \mathit{owl{:}allValuesFrom}, \mathit{NotGRID} \sqcup \bigsqcup_{(D_i,d)\in H} d\rangle \text{ for } 1 \le i \le n \tag{11.4}$$

$$\langle D_i, \mathit{rdfs{:}subClassOf}, \beta_i\rangle, \langle \beta_i, \mathit{owl{:}onProperty}, \mathit{rdf{:}type}\rangle, \langle \beta_i, \mathit{owl{:}allValuesFrom}, \mathit{NotGRID} \sqcup \bigsqcup_{(D_i,d)\in V} d\rangle \text{ for } 1 \le i \le n \tag{11.5}$$

$$\langle \mathit{GRID}, \mathit{owl{:}someValuesFrom}, \mathit{GRID}\rangle \tag{11.6}$$

$$\langle \mathit{GRID}, \mathit{owl{:}onProperty}, \mathit{owl{:}allValuesFrom}\rangle \tag{11.7}$$

$$\langle \mathit{GRID}, \mathit{owl{:}onProperty}, \mathit{rdf{:}type}\rangle \tag{11.8}$$

$$\langle \mathit{GRID}, \mathit{rdfs{:}subClassOf}, \mathit{owl{:}allValuesFrom}\rangle \tag{11.9}$$

$$\langle \mathit{rdf{:}type}, \mathit{owl{:}sameAs}, \mathit{owl{:}onProperty}\rangle \tag{11.10}$$

$$\langle a_{0,0}, \mathit{rdf{:}type}, \mathit{GRID} \sqcap D_0\rangle \tag{11.11}$$

**Lemma 11.1.2.** *A $\mathcal{D}$-tiling exists if and only if $KB_\mathcal{D}$ is satisfiable.*

*Proof.* ($\Rightarrow$) For a $\mathcal{D}$-tiling $t$, let $I$ be an interpretation depicted in Figure 11.1, with $\mathrm{CEXT}^I(\mathit{GRID}^I) = \{a_{i,j}\}$ and $\mathrm{CEXT}^I(D_k^I) = \{a_{i,j} \mid t(i,j) = D_k\}$, for $i, j \ge 0$ and $1 \le k \le n$. The triples (11.3)–(11.5) encode the compatibility relations of $\mathcal{D}$ (including *NotGRID* into (11.3) and (11.4) ensures that compatibility is enforced only among instances of *GRID*). It is easy to see that $I$ is a model of $KB_\mathcal{D}$.

($\Leftarrow$) Let $I$ be a model of $KB_\mathcal{D}$. An excerpt of $I$ is shown in Figure 11.1, in which a triple $\langle s, p, o\rangle$ is represented as an arc pointing from the node $s$ to the node $o$, whereas $p$ is encoded by the line type according to the legend. To refer easily to arcs, we assign them labels $t_i$, $h_i$, and $v_i$ (these do not correspond to $p$). For example, the arc $s_1$ represents the triple $\langle a_{0,0}, \mathit{rdf{:}type}, \mathit{owl{:}allValuesFrom}\rangle$. Due to (11.10), *rdf:type* and *owl:onProperty* are synonyms, so $s_1$ also represents the triple $\langle a_{0,0}, \mathit{owl{:}onProperty}, \mathit{owl{:}allValuesFrom}\rangle$. By an abuse of notation, we do not distinguish between the symbols and their interpretations.

Due to (11.11), $a_{0,0}$ is linked by $t_1$ to *GRID*. Due to (11.6), (11.7), and (11.8), $a_{0,0}$ is linked to $a_{0,1}$ and $a_{1,0}$ through $h_1$ and $v_1$, respectively, and $a_{0,1}$ and $a_{1,0}$ are in the

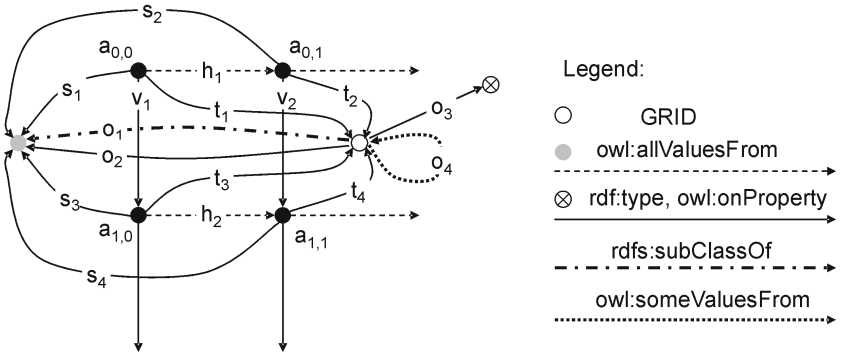



Figure 11.1: Grid Structure in a Model of $KB_{\mathcal{D}}$

concept extension of *GRID* by $t_2$ and $t_3$, respectively. Due to (11.6) and (11.7), $a_{1,0}$ is linked by $h_2$ to $a_{1,1}$, and by $t_4$ to *GRID*. Finally, by (11.9), all $a_{i,j}$ are in the concept extension of *owl:allValuesFrom*; that is, all $a_{i,j}$ have an $s_l$ arc to it.

Consider now the arcs at the node $a_{1,0}$. The arc $s_3$ can, due to (11.10), be read as $\langle a_{1,0}, \mathit{owl:onProperty}, \mathit{owl:allValuesFrom}\rangle$. By applying Item 11 of Table 11.1 for $x = a_{1,0}$ and $y = a_{1,1}$, we see that, if $w$ is in the concept extension of $a_{1,0}$ and it is connected via $p = \mathit{owl:allValuesFrom}$ to some $z$, then $z$ must be in the concept extension of $a_{1,1}$. By setting $w = a_{0,0}$ due to $v_1$ and $z = a_{0,1}$ due to $h_1$, we get that $a_{0,1}$ is in the concept extension of $a_{1,1}$. Hence, $a_{0,1}$ is connected to $a_{1,1}$ by $v_2$, so $a_{0,0}$, $a_{0,1}$, $a_{1,0}$, and $a_{1,1}$ are arranged in a two-dimensional grid, which continues indefinitely due to (11.6)–(11.8).

A node $a_{i,j}$ in $I$ is allowed to have multiple *owl:allValuesFrom* and *rdf:type* successors, and all $a_{i,j}$ need not be distinct, so $I$ need not be a two-dimensional grid. However, a two-dimensional grid can easily be extracted from $I$: one can choose any *owl:allValuesFrom* successor $a_{i,j+1}$ and any *rdf:type* successor $a_{i+1,j}$ of $a_{i,j}$, as well as any *owl:allValuesFrom* successor $a_{i+1,j+1}$ of $a_{i+1,j}$. Regardless of the choices, $a_{i,j+1}$ is always connected to $a_{i+1,j+1}$ by *rdf:type*, so $a_{i,j}$, $a_{i,j+1}$, $a_{i+1,j}$, and $a_{i+1,j+1}$ are connected in a grid-like manner.

Hence, $I$ contains a two-dimensional infinite grid in which *owl:allValuesFrom* are horizontal, and *rdf:type* are vertical arcs. The triples (11.1)–(11.5) ensure that each grid node is assigned a single domino type that corresponds to the compatibility relations $H$ and $V$ of $\mathcal{D}$, so a $\mathcal{D}$-tiling can easily be constructed from $I$. □

We briefly comment on an important issue in the proof of Lemma 11.1.2. Namely, the main difficulty in the reduction is to ensure that each member of the *GRID* concept has an *owl:onProperty* arc to the *owl:allValuesFrom* node, so that we can apply Item 11 of Table 11.1. In the case of OWL-Full, this might be done using nominals, by adding to the *GRID* concept an existential restriction on *owl:onProperty* to the nominal *owl:allValuesFrom*. However, to obtain a more general result, we refrain from

using nominals, and employ the following trick instead. By (11.9) we say that *GRID* is a subclass of *owl:allValuesFrom*, which ensures that every member of *GRID* has an *rdf:type* arc to the *owl:allValuesFrom* node. Furthermore, by (11.10) we say that *rdf:type* and *owl:allValuesFrom* are synonyms. Now these two axioms are sufficient to obtain the necessary *owl:onProperty* arc pointing to the *owl:allValuesFrom* node for each member of the *GRID* concept.

Undecidability of $\mathcal{ALC}$-Full follows as an immediate consequence of Lemma 11.1.2 and the known undecidability result for the domino tiling problem [20]:

**Theorem 11.1.3.** *Checking satisfiability of an $\mathcal{ALC}$-Full knowledge base KB is undecidable.*

## 11.2 Extending DLs with Decidable Metamodeling

The proof of Lemma 11.1.2 reveals the causes for the undecidability of metamodeling in OWL-Full. Namely, this logic not only allows treating concepts as individuals, but it also allows mixing logical and metalogical symbols, thus exposing its modeling primitives as individuals. We exploited this in axioms (11.5) and (11.6) of $KB_{\mathcal{D}}$, by stating an existential restriction on *owl:allValuesFrom* and *rdf:type* symbols, thus affecting their semantics. One would easily agree that tampering with the semantics of the ontology language is hardly desirable in practice, so in this section we present two alternative semantics for metamodeling. Due to certain technical problems, we add metamodeling to $\mathcal{ALCHIQ}(\mathbf{D})$ first, and consider transitive roles subsequently.

### 11.2.1 Metamodeling Semantics for $\mathcal{ALCHIQ}(\mathbf{D})$

To allow for metamodeling, we extend the syntax of description logics (Definitions 3.1.1 and 3.2.3) such that $N_C$, $N_{I_a}$, $N_{R_a}$, and $N_{R_c}$ are not necessarily disjoint, and are contained in a set of *names* $N$, for which $n \in N$ implies $\mathsf{Inv}(n) \in N$. Hence, the same symbol can be used to denote a concept, an abstract individual, an abstract role, and a concrete role. This causes problems for axioms of the form $A \sqsubseteq B$: from the axiom alone, it is not clear whether the inclusion refers to concept, to abstract role, or to concrete role interpretations of $A$ and $B$. To distinguish these three types of inclusions, we write $A \sqsubseteq_a B$ for abstract role inclusion, $A \sqsubseteq_c B$ for concrete role inclusion, and $A \sqsubseteq_n B$ for concept inclusion. To simplify the presentation, we do not consider axioms of the form $A \equiv B$, as they are simply shortcuts for $A \sqsubseteq_n B$ and $B \sqsubseteq_n A$.

To avoid technical complications, we assume that the set of names $N$, the set of concrete predicates $\Phi_{\mathbf{D}}$, and the set of concrete individuals $R_{I_c}$ are pair-wise disjoint. We believe that this is not a practically important restriction: we cannot think of an example in which treating concrete predicates as individuals, or mixing abstract and concrete individuals might be required. This also allows us to syntactically distinguish complex concepts: because $A \in N$ and $d \in \Phi_{\mathbf{D}}$, it is clear that $\exists R.A$ denotes existential quantification over the abstract domain, whereas $\exists T.d$ denotes existential quantifica-

tion over the concrete domain. Similarly, this allows us to syntactically distinguish abstract and concrete ABox axioms, such as $S(a, b)$ and $T(a, b^{\mathsf{c}})$.

For thus modified syntax, we now define the *contextual* semantics of metamodeling, whose name reflects that a symbol in the knowledge base is interpreted as a concept, a role, or an individual according to the syntactic context where it occurs. We use this semantics mainly for comparison with the HiLog semantics, which we introduce later.

**Definition 11.2.1** (Contextual Semantics)**.** *Let KB be an $\mathcal{ALCHIQ}(\mathbf{D})$ knowledge base over an admissible concrete domain $\mathbf{D}$. A $\pi$-interpretation $I = (\triangle^I, \cdot^I, C_n^I, R_a^I, R_c^I)$ is a 5-tuple where $\triangle^I$ is a* domain set*, $\cdot^I : N \to \triangle^I$ is a* symbol interpretation function*, $C_n^I : N \to 2^{\triangle^I}$ is an* atomic concept extension function*, $R_a^I : N \to 2^{\triangle^I \times \triangle^I}$ is an* abstract role extension function*, and $R_c^I : N \to 2^{\triangle^I \times \triangle_{\mathbf{D}}}$ is a* concrete role extension function*.*

*The function $C_n^I$ is extended to concepts as specified in Table 11.2, upper left section, where symbols are interpreted* contextually*—that is, depending on their syntactic position. A $\pi$-interpretation $I$ is a $\pi$-model of KB if it satisfies all conditions from Table 11.2, lower left section. The notions of $\pi$-satisfiability, $\pi$-unsatisfiability, and $\pi$-entailment (written $\models_\pi$) are defined as usual.*

The contextual semantics is essentially equivalent to the contextual semantics from [33], and to the standard first-order semantics. Namely, in a first-order formula, the role of a symbol can be inferred from the place at which the symbol occurs in a formula, so the sets of constant, function, and predicate symbols do not need to be pairwise disjoint. Hence, $KB$ is $\pi$-satisfiable if and only if $\pi(KB)$ is (first-order) satisfiable, which can be decided, for example, using the algorithms from Chapters 5 and 6. Note that, as mentioned in Section 2.5, in basic superposition we encode literals as $\mathcal{E}$-terms. For a correct encoding under the contextual semantics, we additionally use a separate sort for the $\mathcal{E}$-general function symbols obtained by encoding predicate symbols.

We now define the *HiLog* semantics for metamodeling, which follows the ideas of OWL-Full more closely.

**Definition 11.2.2** (HiLog Semantics)**.** *Let KB be an $\mathcal{ALCHIQ}(\mathbf{D})$ knowledge base over an admissible concrete domain $\mathbf{D}$. A $\nu$-interpretation $I = (\triangle^I, \cdot^I, C_n^I, R_a^I, R_c^I)$ is a 5-tuple where $\triangle^I$ is a* domain set*, $\cdot^I : N \to \triangle^I$ is a* symbol interpretation function*, $C_n^I : \triangle^I \to 2^{\triangle^I}$ is an* atomic concept extension function*, $R_a^I : \triangle^I \to 2^{\triangle^I \times \triangle^I}$ is an* abstract role extension function*, and $R_c^I : \triangle^I \to 2^{\triangle^I \times \triangle_{\mathbf{D}}}$ is a* concrete role extension function*.*

*The extension of the function $C_n^I$ to concepts, and the interpretation of axioms are specified in Table 11.2, right section. The notions of $\nu$-satisfiability, $\nu$-unsatisfiability, and $\nu$-entailment (written $\models_\nu$) are defined as usual.*

We briefly discuss the essential difference between these two semantics. Consider the knowledge base consisting only of one axiom $a(a)$, where the symbol $a$ is used as an individual and as a concept. A $\pi$-model of such a knowledge base is depicted on the left-hand side of Figure 11.2, where both the individual interpretation $\cdot^I$ and the concept

Table 11.2: Two Semantics for $\mathcal{SHIQ}(\mathbf{D})$ with Metamodeling

| $\pi$-semantics | | $\nu$-semantics |
|---|---|---|
| Extending $C_n^I$ to concepts | | $C_n^I$ and the interpretation of axioms are obtained from the ones for $\pi$-semantics by applying the following changes: $C_n^I(A) \rightsquigarrow C_n^I(A^I)$, $R_a^I(\mathsf{Inv}(S)) \rightsquigarrow R_a^I(\mathsf{Inv}(S)^I)$, $R_a^I(S) \rightsquigarrow R_a^I(S^I)$, $R_a^I(T) \rightsquigarrow R_a^I(T^I)$, $R_c^I(S) \rightsquigarrow R_c^I(S^I)$, $R_c^I(T) \rightsquigarrow R_c^I(T^I)$ |
| $A$ | $C_n^I(A) \subseteq \triangle^I$ | |
| $\neg D$ | $\triangle^I \setminus C_n^I(D)$ | |
| $D_1 \sqcap D_2$ | $C_n^I(D_1) \cap C_n^I(D_2)$ | |
| $D_1 \sqcup D_2$ | $C_n^I(D_1) \cup C_n^I(D_2)$ | |
| $\exists S.D$ | $\{x \mid (x,y) \in R_a^I(S) \wedge y \in C_n^I(D)\}$ | |
| $\forall S.D$ | $\{x \mid (x,y) \in R_a^I(S) \rightarrow y \in C_n^I(D)\}$ | |
| $\leq n\, S.D$ | $\{x \mid \sharp\{y \mid (x,y) \in R_a^I(S) \wedge y \in C_n^I(D)\} \leq n\}$ | |
| $\geq n\, S.D$ | $\{x \mid \sharp\{y \mid (x,y) \in R_a^I(S) \wedge y \in C_n^I(D)\} \geq n\}$ | |
| $(\forall T_1, \ldots, T_m.d)^I$ | $\{x \mid \forall y_1, \ldots, y_m : \bigwedge(x, y_i) \in R_c^I(T_i) \rightarrow (y_1, \ldots, y_m) \in d^{\mathbf{D}}\}$ | |
| $(\exists T_1, \ldots, T_m.d)^I$ | $\{x \mid \exists y_1, \ldots, y_m : \bigwedge(x, y_i) \in R_c^I(T_i) \wedge (y_1, \ldots, y_m) \in d^{\mathbf{D}}\}$ | |
| $(\leq n\, T)^I$ | $\{x \mid \sharp\{y \mid (x,y) \in R_c^I(T)\} \leq n\}$ | |
| $(\geq n\, T)^I$ | $\{x \mid \sharp\{y \mid (x,y) \in R_c^I(T)\} \geq n\}$ | |
| Interpretation of axioms | | |
| | $R_a^I(S) = R_a^I(\mathsf{Inv}(S))^-$ | |
| $S \sqsubseteq_a T$ | $R_a^I(S) \subseteq R_a^I(T)$ | |
| $S \sqsubseteq_c T$ | $R_c^I(S) \subseteq R_c^I(T)$ | |
| $D_1 \sqsubseteq_n D_2$ | $C_n^I(D_1) \subseteq C_n^I(D_2)$ | |
| $\mathsf{Trans}(S)$ | $R_a^I(S)^+ \subseteq R_a^I(S)$ | |
| $D(a)$ | $a^I \in C_n^I(D)$ | |
| $S(a,b)$ | $(a^I, b^I) \in R_a^I(S)$ | |
| $\neg S(a,b)$ | $(a^I, b^I) \notin R_a^I(S)$ | |
| $T(a, b^{\mathsf{c}})$ | $(a^I, (b^{\mathsf{c}})^I) \in R_c^I(T)$ | |
| $\neg T(a, b^{\mathsf{c}})$ | $(a^I, (b^{\mathsf{c}})^I) \notin R_c^I(T)$ | |
| $a^{(\mathsf{c})} \approx b^{(\mathsf{c})}$ | $(a^{(\mathsf{c})})^I = (b^{(\mathsf{c})})^I$ | |
| $a^{(\mathsf{c})} \not\approx b^{(\mathsf{c})}$ | $(a^{(\mathsf{c})})^I \neq (b^{(\mathsf{c})})^I$ | |

**Note:** $\sharp N$ is the number of elements in $N$; $S^+$ is the transitive closure of $S$; and $S^-$ is the inverse relation of $S$.

interpretation $C_n^I$ are assigned directly to the symbol $a$. On the contrary, a $\nu$-model of the knowledge base is depicted on the right-hand side of Figure 11.2. There, the individual interpretation $\cdot^I$ assigns the domain individual $x$ to the symbol $a$; however, the concept interpretation is not assigned to $a$, but to the domain individual $x$. We discuss the practical consequences of such a definition on entailment in Section 11.3.

Since our algorithms are based on resolution calculi, we present an alternative but equivalent definition of $\nu$-models by translation into first-order logic. This translation follows the principles of transforming HiLog formulae into first-order formulae: it reifies concept, abstract, and concrete role symbols into constants, and represents functions $C_n^I$, $R_a^I$, and $R_c^I$ explicitly by predicates isa, arole, and crole, respectively.

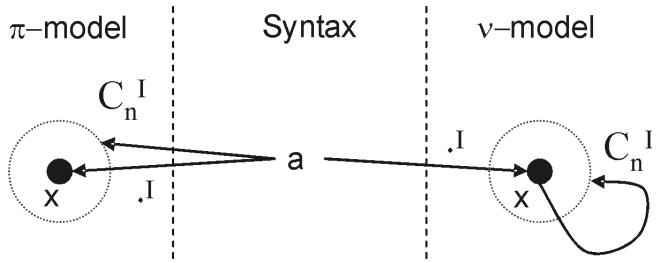



Figure 11.2: $\pi$- and $\nu$-models of the Example Knowledge Base

**Definition 11.2.3.** *For an $\mathcal{ALCHIQ}(\mathbf{D})$ knowledge base KB, let $\nu(KB)$ be the translation of KB into first-order formulae, as specified in Table 11.3, where* $\mathsf{isa}$ *is a binary predicate with signature* $\mathsf{a} \times \mathsf{a}$*,* $\mathsf{arole}$ *is a ternary predicate with signature* $\mathsf{a} \times \mathsf{a} \times \mathsf{a}$*, and* $\mathsf{crole}$ *is a ternary predicate with signature* $\mathsf{a} \times \mathsf{a} \times \mathsf{c}$*.*

**Lemma 11.2.4.** *For an $\mathcal{ALCHIQ}(\mathbf{D})$ knowledge base KB, KB is $\nu$-satisfiable if and only if a first-order model of $\nu(KB)$ exists.*

*Proof.* For the ($\Rightarrow$) direction, let $I_\nu$ be a $\nu$-model of $KB$. We construct a first-order interpretation $I$ for $\nu(KB)$ by setting $\triangle^I = \triangle^{I_\nu}$, $a^I = a^{I_\nu}$, $(x, y) \in \mathsf{isa}^I$ if $y \in C_n^{I_\nu}(x)$, $(x, y, z) \in \mathsf{arole}^I$ if $(y, z) \in R_a^{I_\nu}(x)$, and $(x, y, z) \in \mathsf{crole}^I$ if $(y, z) \in R_c^{I_\nu}(x)$. By induction on the structure of formulae in $\nu(KB)$, one can easily show that $I$ is a model of $\nu(KB)$. The ($\Leftarrow$) direction is similar. □

In [106], it was stressed that the semantics of OWL-Full and OWL-DL coincide, provided that resources defining modeling primitives are not used in the knowledge base axioms, and that the sets of concept, role, and individual symbols are disjoint. Similarly, it is possible to show that the HiLog semantics coincides with the OWL-Full semantics if resources defining modeling primitives are not used in knowledge base axioms. The formal proof for this is simple, but it is lengthy due to the very complex semantics of OWL-Full, so we omit it for the sake of brevity.

### 11.2.2 Deciding $\nu$-Satisfiability

We now show that $\nu$-satisfiability can be decided by extending the algorithms from Chapters 5 and 6, without increasing the complexity of reasoning.

It is difficult to ensure termination of direct saturation of closures obtained by structural transformation of $\nu(KB)$, since such closures contain unmarked constants corresponding to names of concepts and roles. Consider, for example, the following set of closures obtained during saturation:

Table 11.3: Semantics of Metamodeling by Mapping into First-Order Logic

| Mapping Concepts to FOL | |
|---|---|
| $\nu_y(A, X)$ | $= \mathsf{isa}(A, X)$ |
| $\nu_y(\neg D, X)$ | $= \neg\nu_y(D, X)$ |
| $\nu_y(D_1 \sqcap D_2, X)$ | $= \nu_y(D_1, X) \wedge \nu_y(D_2, X)$ |
| $\nu_y(D_1 \sqcup D_2, X)$ | $= \nu_y(D_1, X) \vee \nu_y(D_2, X)$ |
| $\nu_y(\forall R.D, X)$ | $= \forall y : \mathsf{arole}(R, X, y) \rightarrow \nu_x(D, y)$ |
| $\nu_y(\exists R.D, X)$ | $= \exists y : \mathsf{arole}(R, X, y) \wedge \nu_x(D, y)$ |
| $\nu_y(\leq n\, R.D, X)$ | $= \forall y_1, \ldots, y_{n+1} : \bigwedge_{i=1}^{n+1} [\mathsf{arole}(R, X, y_i) \wedge \nu_x(D, y_i)] \rightarrow \bigvee_{i=1}^{n+1} \bigvee_{j=i+1}^{n+1} y_i \approx y_j$ |
| $\nu_y(\geq n\, R.D, X)$ | $= \exists y_1, \ldots, y_n : \bigwedge_{i=1}^{n} [\mathsf{arole}(R, X, y_i) \wedge \nu_x(D, y_i)] \wedge \bigwedge_{i=1}^{n} \bigwedge_{j=i+1}^{n} y_i \not\approx y_j$ |
| $\nu_y(\forall T_1, \ldots, T_m.d, X)$ | $= \forall y_1^{\mathsf{c}}, \ldots, y_m^{\mathsf{c}} : \bigwedge_{i=1}^{m} \mathsf{crole}(T_i, X, y_i^{\mathsf{c}}) \rightarrow d(y_1^{\mathsf{c}}, \ldots, y_m^{\mathsf{c}})$ |
| $\nu_y(\exists T_1, \ldots, T_m.d, X)$ | $= \exists y_1^{\mathsf{c}}, \ldots, y_m^{\mathsf{c}} : \bigwedge_{i=1}^{m} \mathsf{crole}(T_i, X, y_i^{\mathsf{c}}) \wedge d(y_1^{\mathsf{c}}, \ldots, y_m^{\mathsf{c}})$ |
| $\nu_y(\leq n\, T, X)$ | $= \forall y_1^{\mathsf{c}}, \ldots, y_{n+1}^{\mathsf{c}} : \bigwedge_{i=1}^{n+1} \mathsf{crole}(T, X, y_i^{\mathsf{c}}) \rightarrow \bigvee_{i=1}^{n+1} \bigvee_{j=i+1}^{n+1} y_i^{\mathsf{c}} \approx_{\mathbf{D}} y_j^{\mathsf{c}}$ |
| $\nu_y(\geq n\, T, X)$ | $= \exists y_1^{\mathsf{c}}, \ldots, y_n^{\mathsf{c}} : \bigwedge_{i=1}^{n} \mathsf{crole}(T, X, y_i^{\mathsf{c}}) \wedge \bigwedge_{i=1}^{n} \bigwedge_{j=i+1}^{n} y_i^{\mathsf{c}} \not\approx_{\mathbf{D}} y_j^{\mathsf{c}}$ |
| **Mapping Axioms to FOL** | |
| $\nu(R \sqsubseteq_a S)$ | $= \forall x, y : \mathsf{arole}(R, x, y) \rightarrow \mathsf{arole}(S, x, y)$ |
| $\nu(T \sqsubseteq_c U)$ | $= \forall x, y : \mathsf{crole}(T, x, y) \rightarrow \mathsf{crole}(U, x, y)$ |
| $\nu(D_1 \sqsubseteq_n D_2)$ | $= \forall x : \nu_y(D_1, x) \rightarrow \nu_y(D_2, x)$ |
| $\nu((\neg)D(a))$ | $= (\neg)\nu_y(D, a)$ |
| $\nu((\neg)R(a, b))$ | $= (\neg)\mathsf{arole}(R, a, b)$ |
| $\nu((\neg)T(a, b^{\mathsf{c}}))$ | $= (\neg)\mathsf{crole}(T, a, b^{\mathsf{c}})$ |
| $\nu(a \circ b)$ | $= a \circ b$ for $\circ \in \{\approx, \not\approx\}$ |
| $\nu(a^{\mathsf{c}} \circ b^{\mathsf{c}})$ | $= a^{\mathsf{c}} \circ_{\mathbf{D}} b^{\mathsf{c}}$ for $\circ \in \{\approx, \not\approx\}$ |
| **Mapping $KB$ to FOL** | |
| $\nu(R)$ | $= \forall x, y : \mathsf{arole}(R, x, y) \leftrightarrow \mathsf{arole}(R^-, y, x)$ |
| $\nu(KB_{\mathcal{R}})$ | $= \bigwedge_{\alpha \in KB_{\mathcal{R}}} \nu(\alpha) \wedge \bigwedge_{R \in N} \nu(R)$ |
| $\nu(KB_{\mathcal{T}})$ | $= \bigwedge_{\alpha \in KB_{\mathcal{T}}} \nu(\alpha)$ |
| $\nu(KB_{\mathcal{A}})$ | $= \bigwedge_{\alpha \in KB_{\mathcal{A}}} \nu(\alpha)$ |
| $\nu(KB)$ | $= \nu(KB_{\mathcal{R}}) \wedge \nu(KB_{\mathcal{T}}) \wedge \nu(KB_{\mathcal{A}})$ |

Notes:
($i$): $X$ is a meta-variable and is substituted by the actual term;
($ii$): $\nu_x$ is obtained from $\nu_y$ by simultaneously substituting in the definition all $y_{(i)}$ with $x_{(i)}$, $\nu_y$ with $\nu_x$, and vice versa.

$$\mathsf{isa}(C, x) \tag{11.12}$$

$$\mathsf{isa}(D, x) \tag{11.13}$$

$$C \approx C' \vee D \approx D' \tag{11.14}$$

Superposition of (11.14) into (11.12) and (11.13) yields (11.15). The problem with this closure is that it contains two variables. Further inferences with it might yield a closure with even more variables, so we cannot establish a bound on closure length.

$$\mathsf{isa}(C', x) \vee \mathsf{isa}(D', x') \tag{11.15}$$

We solve this problems by modifying the preprocessing step. A closure such as (11.12) is decomposed into closures $\neg\mathcal{O}_C(z) \vee \mathsf{isa}(z, x)$ and $\mathcal{O}_C(C)$. Furthermore, the literal $\neg\mathcal{O}_C(z)$ is selected, so the first resolution inference produces $\mathsf{isa}([C], x)$. The constant $C$ now occurs at substitution position, so superposition into it is not necessary. Moreover, we decompose a ground closure $\mathcal{O}_P(\langle Q \rangle) \vee C \cdot \rho$ into $\neg q_{P,Q} \vee \mathcal{O}_P(Q)$ and $q_{P,Q} \vee C \cdot \rho$ to prevent closures of types other than 8 to contain arbitrary ground literals.

**Definition 11.2.5.** *For an $\mathcal{ALCHIQ}(\mathbf{D})$-closure $C$, let $\mathsf{Cls}_\nu$ be defined as follows:*

$$\mathsf{Cls}_\nu(C) = \bigcup_{D \in \mathsf{Def}(C)} \mathsf{Cls}(\forall x : \nu_y(D, x))$$

*Let $\zeta(C)$ be the closure obtained from $C$ by replacing all literals according to the following table, where the predicate $\mathcal{O}_P$ is globally unique for the predicate symbol $P$, $z_P$ is a new variable globally unique for $P$, and $u$ and $v$ are arbitrary terms:*

$$\begin{array}{lcl} \mathsf{isa}(P, u) & \rightsquigarrow & \mathsf{isa}(z_P, u) \vee \neg\mathcal{O}_P(z_P) \\ \mathsf{arole}(P, u, v) & \rightsquigarrow & \mathsf{arole}(z_P, u, v) \vee \neg\mathcal{O}_P(z_P) \\ \mathsf{crole}(P, u, v^{\mathsf{c}}) & \rightsquigarrow & \mathsf{crole}(z_P, u, v^{\mathsf{c}}) \vee \neg\mathcal{O}_P(z_P) \end{array}$$

*The operator $\zeta$ is extended to a set of closures by applying it to each member of the set. For an extensionally reduced $\mathcal{ALCHIQ}(\mathbf{D})$ knowledge base $KB$, $\Xi_\nu(KB)$ is the smallest set of closures satisfying the following conditions:*

- *For each name $R \in N$, $\zeta(\mathsf{Cls}(\nu(R))) \subseteq \Xi_\nu(KB)$;*
- *For each RBox or ABox axiom $\alpha$ in $KB$, $\zeta(\mathsf{Cls}(\nu(\alpha))) \subseteq \Xi_\nu(KB)$;*
- *For each TBox axiom $C \sqsubseteq_n D$ in $KB$, $\zeta(\mathsf{Cls}_\nu(\neg C \sqcup D)) \subseteq \Xi_\nu(KB)$;*
- *For each predicate $\mathcal{O}_P$ introduced by $\zeta$, $\mathcal{O}_P(P) \in \Xi_\nu(KB)$.*

*If $KB$ is not extensionally reduced, then $\Xi_\nu(KB) = \Xi_\nu(KB')$, where $KB'$ is an extensionally reduced knowledge base obtained from $KB$ as explained in Section 3.1.*

It is easy to see that $\Xi_\nu(KB)$ is satisfiable if and only if $KB$ is $\nu$-satisfiable.

**Lemma 11.2.6.** *Let $KB$ be an $\mathcal{ALCHIQ}(\mathbf{D})$ knowledge base. Then $KB$ is $\nu$-satisfiable if and only if $\Xi_\nu(KB)$ is satisfiable. Furthermore, $\Xi_\nu(KB)$ can be computed in time polynomial in $|KB|$, for unary coding of numbers in input.*

*Proof.* By Lemma 11.2.4, $KB$ is $\nu$-satisfiable if and only if $\nu(KB)$ is satisfiable. $\Xi_\nu(KB)$ is obtained from $\nu(KB)$ by applying structural transformation, which is known not to affect satisfiability, and by applying decomposition to literals $\mathsf{isa}(P, u)$, $\mathsf{arole}(P, u, v)$, and $\mathsf{crole}(P, u, v^{\mathsf{c}})$, which does by affect satisfiability by Theorem 5.4.4. □

**Definition 11.2.7.** *Let $\mathcal{BS}^{\mathbf{D},+}_\nu$ be the $\mathcal{BS}^{\mathbf{D}}$ calculus parameterized as follows:*

- *The $\mathcal{E}$-term ordering $\succ$ is a lexicographic path ordering induced by a total precedence $>$ over function, constant, and predicate symbols such that, for any function symbol $f$, constant symbol $c$, predicate symbol $P$, and a propositional symbol $q_{P,Q}$, we have $f > c > P > q_{P,Q} > \mathsf{T}$.*
- *If a closure contains a literal of the form $\neg\mathcal{O}_C(z)$, then all such literals are selected; otherwise, all negative* arole *and* crole *literals are selected.*
- *Inference conclusions, whenever possible, are decomposed according to the following table, for an arbitrary term $t$:*

$$D \cdot \rho \vee \mathsf{arole}([R], [t], [f(t)]) \rightsquigarrow \begin{array}{c} D \cdot \rho \vee Q_{R,f}([t]) \\ \neg Q_{R,f}(x) \vee \mathsf{arole}([R], x, [f(x)]) \end{array}$$

$$D \cdot \rho \vee \mathsf{arole}([R], [f(x)], x) \rightsquigarrow \begin{array}{c} D \cdot \rho \vee Q_{\mathsf{Inv}(R),f}(x) \\ \neg Q_{\mathsf{Inv}(R),f}(x) \vee \mathsf{arole}([R], [f(x)], x) \end{array}$$

$$\mathcal{O}_P(\langle Q \rangle) \vee C \cdot \rho \rightsquigarrow \begin{array}{c} C \cdot \rho \vee q_{P,Q} \\ \neg q_{P,Q} \vee \mathcal{O}_P(Q) \end{array}$$

We define *$\nu$-$\mathcal{ALCHIQ}(\mathbf{D})$-closures* to be of the same form as $\mathcal{ALCHIQ}(\mathbf{D})$-closures from Tables 5.1 and 6.1 with the following differences:

- Conditions (*iii*)–(*vi*) are dropped;
- Atoms of the form $C(t)$ take the form $\mathsf{isa}([C], t)$;
- Atoms of the form $R(u, v)$ take the form $\mathsf{arole}([R], u, v)$;
- Atoms of the form $T(u, v^{\mathsf{c}})$ take the form $\mathsf{crole}([T], u, v^{\mathsf{c}})$;
- All closures can contain a disjunction of the form $\mathbf{q} = \bigvee(\neg)q_{P,Q}$.

Note that closures in $\Xi_\nu(KB)$ are technically not $\nu$-$\mathcal{ALCHIQ}(\mathbf{D})$-closures: instead of literals of the form $(\neg)\mathsf{isa}([C], x)$, they contain $(\neg)\mathsf{isa}(z, x) \vee \neg\mathcal{O}_C(z)$. We call such closures *initial closures*.

**Lemma 11.2.8.** *Let $\Xi_\nu(KB) = N_0, \ldots, N_i \cup \{C\}$ be a $\mathcal{BS}_\nu^{\mathbf{D},+}$-derivation, where $C$ is the conclusion derived from premises in $N_i$. Then, $C$ is either a $\nu$-$\mathcal{ALCHIQ}(\mathbf{D})$-closure, or it is redundant in $N_i$.*

*Proof.* Due to decomposition, it is obvious that the following property (*) holds: positive literals of the form $\mathcal{O}_P(Q)$ occur only in unit closures, or in closures of the form $\mathcal{O}_P(Q) \vee \neg q_{P,Q}$.

Now consider an inference with an initial closure $C$. Since $C$ contains literals of the form $\neg\mathcal{O}_P(z)$, and all such literals are selected, $C$ can participate only in a hyperresolution inference on all $\neg\mathcal{O}_P(z_i)$. Because of (*), the inference instantiates all variables $z_i$, and the conclusion is a $\nu$-$\mathcal{ALCHIQ}(\mathbf{D})$-closure.

Due to the $\mathcal{E}$-term ordering of $\mathcal{BS}_\nu^{\mathbf{D},+}$, only a literal with a term of the maximum depth can be maximal in a $\nu$-$\mathcal{ALCHIQ}(\mathbf{D})$-closure. Also, the $\mathcal{E}$-term ordering ensures that a propositional letter $q_{P,Q}$ is not maximal if a closure contains some other literal. Finally, appropriate inferences are eligible for decomposition in the same way as in Theorem 5.4.8. Hence, the claim of this lemma can be shown in the same way as in the proofs of Lemma 5.3.6, Theorem 5.4.8, and Lemma 6.2.1. □

**Theorem 11.2.9.** *Let $KB$ be an $\mathcal{ALCHIQ}(\mathbf{D})$ knowledge base, defined over an admissible concrete domain $\mathbf{D}$, for which $\mathbf{D}$-satisfiability of finite conjunctions over $\Phi_\mathbf{D}$ can be decided in deterministic exponential time. Then, saturation of $\Xi_\nu(KB)$ by $\mathcal{BS}_\nu^{\mathbf{D},+}$ with eager application of redundancy elimination rules decides $\nu$-satisfiability of $KB$, and runs in time exponential in $|KB|$, for unary coding of numbers and assuming a bound on the arity of concrete predicates.*

*Proof.* The number of propositional letters $q_{P,Q}$ is quadratic in $|KB|$, and the number of predicates $\mathcal{O}_P$ is linear in $|KB|$. Therefore, it is possible to obtain an exponential bound on the number of closures derived in the same way as in Lemma 5.3.9, Theorem 5.4.8, and Theorem 6.2.5. Since decomposition is sound and complete by Theorem 5.4.4, the claim of this theorem follows. □

Note that $\nu$-semantics reifies concept and roles; however, it is more like $\pi$-semantics, and less like OWL-Full semantics in the way it handles modeling primitives. In $\nu$- and $\pi$-semantics, the modeling primitives are expressed as formulae, so they cannot be tampered with. On the contrary, OWL-Full reifies the modeling primitives as well, so their semantics can be affected by statements in the knowledge base.

### 11.2.3 Metamodeling and Transitivity

Since the algorithm for checking $\nu$-satisfiability of an $\mathcal{ALCHIQ}(\mathbf{D})$ knowledge base $KB$ does not differ essentially from the algorithm for checking $\pi$-satisfiability of $KB$,

one might intuitively expect that allowing transitivity axioms in $KB$ does not cause problems. However, consider the following knowledge base $KB$:

$$\top \sqsubseteq_n \geq 3\, R \tag{11.16}$$

$$R \approx S \tag{11.17}$$

$$\mathsf{Trans}(S) \tag{11.18}$$

Note that $KB$ is a $\mathcal{SHIQ}$ knowledge base: the role $S$ is simple, since it passes the syntactic criterion specified in Definition 3.1.1 (it is neither transitive, nor it has a transitive subrole). However, in any $\nu$-interpretation $I$, the axiom (11.17) ensures that $S^I = T^I = \alpha$. Furthermore, due to (11.18), $R^I_a(\alpha)$ is transitive. Effectively, in (11.16) a transitive role $R$ is used in a number restriction, even though it is syntactically a simple role.

Since equality of roles might be nontrivially entailed by $KB$, identifying simple roles would itself require theorem proving. Therefore, it is difficult, if not impossible, to define simple roles under $\nu$-semantics. In [73], it was shown that transitive roles in number restrictions lead to undecidability, so we get the following result:

**Proposition 11.2.10.** *Checking $\nu$-satisfiability of a $\mathcal{SHIQ}$ knowledge base $KB$ is undecidable.*

Decidability can be regained by using *unique role assumption*. Intuitively, this assumption requires two distinct role symbols to be interpreted as distinct domain individuals. In such a case, simple roles can be defined and checked as usual.

**Definition 11.2.11** (Unique Role Assumption)**.** *A $\mathcal{SHIQ}(\mathbf{D})$ knowledge base $KB$ employs the* unique role assumption *if it contains an axiom $R \not\approx S$, for each two distinct symbols $R$ and $S$ occurring as roles in $KB$.*

It is easy to see that transitivity axioms can be eliminated from knowledge bases employing unique roles assumption using the transformation from Section 5.2, so we thus obtain an algorithm for checking $\nu$-satisfiability of such knowledge bases.

**Theorem 11.2.12.** *Let $KB$ be a $\mathcal{SHIQ}(\mathbf{D})$ knowledge base $KB$ employing the unique role assumption, or containing neither explicit equality statements, nor number restrictions. Then, $KB$ is $\nu$-satisfiable if and only if $\Omega(KB)$ is $\nu$-satisfiable.*

*Proof.* The ($\Rightarrow$) direction of the proof of Theorem 5.2.3 obviously holds in this case as well. For the ($\Leftarrow$) direction, by the lemma assumptions on $KB$, if $\Omega(KB)$ is $\nu$-satisfiable, then it has a $\nu$-model such that $R^{I_\nu} \neq S^{I_\nu}$, for each two distinct symbols $R$ and $S$ occurring as roles in $KB$. Hence, it is possible to construct a $\nu$-model of $KB$ in the same way as in the proof of the ($\Leftarrow$) direction of Theorem 5.2.3. □

## 11.3 Expressivity of Metamodeling

In this section we investigate in which way metamodeling increases the expressivity of the logic. Here we consider only the logical aspects—that is, what kinds of new consequences can be drawn. Our discussion in this section does not consider nonlogical aspects that may be relevant in practice (such as increased search flexibility). The following results are similar to the ones for HiLog [33].

It is easy to see that $\nu$-satisfiability is a strictly stronger notion than $\pi$-satisfiability. Consider the following knowledge base $KB$:[2]

$$Eagle(Harry) \tag{11.19}$$

$$\neg Aquila(Harry) \tag{11.20}$$

$$Eagle \approx Aquila \tag{11.21}$$

Under the contextual semantics, the interpretations of symbols $C$ and $D$ as concepts and as individuals are completely independent, so $KB$ is $\pi$-satisfiable. However, $KB$ is $\nu$-unsatisfiable: in each $\nu$-interpretation, $Eagle^I = Aquila^I = \alpha$, so it cannot be that $Harry^I \in C_n^I(Eagle^I)$ and $Harry^I \notin C_n^I(Aquila^I)$. For the other direction, we have the following lemma:

**Lemma 11.3.1.** *Each $\nu$-satisfiable $\mathcal{ALCHIQ}(\mathbf{D})$ knowledge base is $\pi$-satisfiable.*

*Proof.* Let $I_\nu$ be a $\nu$-model of an $\mathcal{ALCHIQ}(\mathbf{D})$ knowledge base $KB$. We construct from $I_\nu$ a $\pi$-interpretation $I_\pi$ as follows: $\triangle^{I_\pi} = \triangle^{I_\nu}$, $n^{I_\pi} = n^{I_\nu}$, $C_n^{I_\pi}(n) = C_n^{I_\nu}(n^{I_\nu})$, $R_a^{I_\pi}(n) = R_a^{I_\nu}(n^{I_\nu})$, and $R_c^{I_\pi}(n) = R_c^{I_\nu}(n^{I_\nu})$, for each $n \in N$. By a simple induction on the concept structure, it can be shown that, for each concept $X$, $C_n^{I_\pi}(X) = C_n^{I_\nu}(X)$, so $I_\pi$ is a $\pi$-model of $KB$. □

Furthermore, for a knowledge base with unique name assumption or without equality (either explicit or implicit, introduced through number restrictions), $\pi$-satisfiability and $\nu$-satisfiability coincide:

**Lemma 11.3.2.** *Let $KB$ be an $\mathcal{ALCHIQ}(\mathbf{D})$ knowledge base such that it employs unique name assumption, or it contains neither explicit equality statements, nor number restrictions. Then, $KB$ is $\pi$-satisfiable if and only if it is $\nu$-satisfiable.*

*Proof.* The ($\Leftarrow$) direction follows from Lemma 11.3.1. For the ($\Rightarrow$) direction, let $KB$ be $\pi$-satisfiable in some model $I_\pi$. Due to the lemma assumptions, we can assume without loss of generality that $n_i \neq n_j$ implies ${n_i}^{I_\pi} \neq {n_j}^{I_\pi}$, for all for $n_i, n_j \in N$ (in the first case, this is because $KB$ employs unique name assumption, and in the second case this is because $KB$ does not employ equality).

Given such a $\pi$-model $I_\pi$, we construct a $\nu$-interpretation $I_\nu$ by setting $\triangle^{I_\nu} = \triangle^{I_\pi}$, $n^{I_\nu} = n^{I_\pi}$, $C_n^{I_\nu}(n^{I_\nu}) = C_n^{I_\pi}(n)$, $R_a^{I_\nu}(n^{I_\nu}) = R_a^{I_\pi}(n)$, and $R_c^{I_\nu}(n^{I_\nu}) = R_c^{I_\pi}(n)$, where $n \in N$. Furthermore, for all $x \in \triangle^{I_\nu}$ such that there is no $n \in N$ with $x = a^{I_\nu}$, let

[2] *Aquila* is the Latin name for eagle.

$C_n^{I_\nu}(x) = R_a^{I_\nu}(x) = R_c^{I_\nu}(x) = \emptyset$. Since we can assume that different names from $N$ are interpreted as different elements from $\triangle^{I_\nu}$, such a construction defines exactly one value of $C_n^{I_\nu}(x)$, $R_a^{I_\nu}(x)$, and $R_c^{I_\nu}(x)$ for each $x \in \triangle^{I_\nu}$, so $I_\nu$ is correctly defined. By a simple induction on the concept structure, it can be shown that $C_n^{I_\nu}(X) = C_n^{I_\pi}(X)$ for each concept $X$, so $I_\nu$ is a $\nu$-model of $KB$. □

To summarize, $\nu$-semantics derives new consequences only if two symbols can be derived to be equal: for example, from (11.19) and (11.21) it is possible to derive $Aquila(Harry)$. Furthermore, under unique name assumption (which is often used in practice), $\nu$-semantics does not produce any new consequences. This seems to suggest that the benefits of $\nu$-semantics do not outweigh its drawbacks—that it is nonstandard, and that it introduces problems for transitive roles. Moreover, $\pi$-semantics might be sufficient for many practical applications.

However, $\nu$-semantics unlocks its full potential when combined with a language more expressive than OWL. For example, by combining $\nu$-semantics with the Semantic Web Rule Language (SWRL) [69], one can explicitly axiomatize the semantics of metaclasses. Returning to the example from the beginning of this chapter, by (11.22) we state that *Eagle* is a *RedListSpecies*, and by a SWRL rule (11.23) we state that instances of species listed in the Red List are not allowed to be hunted. Note that we use in atom $x(y)$ the variable $x$ at the predicate position. Under $\nu$-semantics this is equivalent to $\mathsf{isa}(x, y)$, but under $\pi$-semantics this would not be possible without leaving the confines of first-order logic. Now from (11.19), (11.22), and (11.23), we can infer $CannotHunt(Harry)$; hence, *RedListSpecies* semantically acts as a metaconcept of the *Eagle* concept.

$$RedListSpecies(Eagle) \tag{11.22}$$

$$RedListSpecies(x) \wedge x(y) \rightarrow CannotHunt(y) \tag{11.23}$$

To summarize, from the logical perspective $\nu$-semantics alone does not make much of a difference, and $\pi$-semantics may be sufficient for numerous applications. However, $\nu$-semantics provides a sound foundation for metamodeling, which, when combined with a more expressive logic, such as SWRL, allows us to precisely axiomatize the interaction between classes and metaclasses. It is easy to see that, if $\nu$-semantics is combined with DL-safe rules from Chapter 9, it yields a decidable formalism. Thus, we believe $\nu$-semantics to be very relevant for future extensions of OWL.

## 11.4 Related Work

The definition of $\nu$-satisfiability given in Section 11.2 is inspired by HiLog [33], a logic in which general terms are allowed to occur in place of function and predicate symbols in formulae. The semantics is defined by interpreting each individual as a member of the interpretation domain, and by assigning a functional and a relational interpretation to domain objects. The authors show that HiLog can be considered "syntactic sugar,"

since each HiLog formula can be encoded into an equisatisfiable first-order formula; the definition of the $\nu$ operator in Table 11.3 closely resembles this encoding. Finally, the authors show that a satisfiable first-order formula without equality is also satisfiable under HiLog semantics.

In [102], the RDFS Model Theory was criticized for allowing infinitely many meta-layers. The authors argue that such a semantics is inadequate for the Semantic Web since (*i*) it does not provide adequate support for inferencing; (*ii*) it allows defining classes that contain themselves, which may potentially lead to paradoxes; and (*iii*) by adding classes, one necessarily introduces objects in the interpretation universe. The authors propose RDFS-FA, a stratified four-level approach, consisting of the meta-language layer, the language layer, the ontology layer, and the instance layer. In [68], similar arguments were used to criticize the semantics of OWL-Full. We follow the principles of RDFS-FA by strictly separating the modeling primitives from the ontology and the instance layers; however, to provide for metamodeling, our definition of $\nu$-semantics merges the ontology and the instance layers. Furthermore, we show that the consequences of (*iii*) actually match the intuition behind metamodeling.

In [149], the authors point out the usefulness of metamodeling in many application domains. They propose separation of modeling layers, which are connected using *spanning instances*. However, the authors do not consider the logical consequences of their approach.

By means of an example very similar to ours, in [132] the author argues for the compelling need of metamodeling in practical knowledge representation systems.

# Part IV

# Practical Considerations

# Chapter 12

# Implementation of KAON2

Whereas the techniques presented in previous chapters provide new theoretical insights into the relationship between description logics and disjunctive datalog, our primary motivation for developing them was to obtain a practical alternative to tableau calculi, capable of handling knowledge bases with reasonably large ABoxes. To validate practicability of our algorithms, we implemented them in a new DL reasoning system KAON2.[1] In this chapter, we describe the implementation of the system, and discuss numerous practical aspects. Namely, the implemented algorithms are of high computational complexity, so it is unrealistic to expect that one can directly cast them into code and obtain an efficient implementation. To avoid unnecessary overhead, we developed several important optimizations. In the following section we present the architecture of the system, followed by a detailed discussion of important system components.

## 12.1 KAON2 Architecture

KAON2 fully implements the satisfiability checking algorithm from Chapter 5, the reduction algorithm from Chapter 7, and the DL-safe rules from Chapter 9. Hence, the supported logic is $\mathcal{SHIQ}$ extended with DL-safe rules. Datatypes are not supported yet because it is currently not clear how to combine concrete domain resolution with optimization techniques, such as magic sets. Furthermore, answering conjunctive queries and metamodeling are not supported because it is currently unclear how to cast the presented algorithms into the setting of disjunctive datalog.

KAON2 was implemented completely in Java 1.5. Figure 12.1 describes its technical architecture. The *Ontology API* provides ontology manipulation tasks, such as adding and retrieving ontology axioms. The API fully supports OWL and the Semantic Web Rule Language (SWRL) at the syntactic level. Several similar APIs already exist, such as OWL API [17] or Jena.[2] However, to obtain an efficient system, we had to exert complete control over the internals of the API, which we could not have achieved

[1] `http://kaon2.semanticweb.org/`
[2] `http://jena.sourceforge.net/`

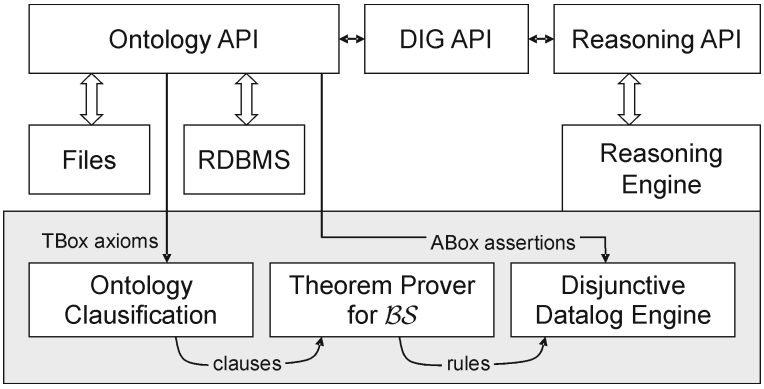



Figure 12.1: KAON2 Architecture

by reusing an existing implementation. Ontologies can be saved in files, using either OWL RDF[3] or OWL XML[4] syntax. Alternatively, ABox assertions can be stored in a relational database (RDBMS): by mapping ontology entities to database tables, KAON2 will query the database on the fly during reasoning.

The *Reasoning API* allows invoking various reasoning tasks, and retrieving their results.

KAON2 can be used as a dynamic library, or as a stand-alone server. In the latter case, the client applications can access KAON2 through either Remote Method Invocation (RMI) protocol or the DL Implementors Group (DIG) API [16]. Thus, the system can be used with Protégé[5] and OilEd[6] ontology editors.

The *Reasoning Engine* is the central component of KAON2, consisting of three subcomponents. The *Ontology Clausification* subcomponent is responsible for translating a $\mathcal{SHIQ}$ knowledge base $KB$ into a set of first-order clauses $\Xi(KB)$. In Section 12.2 we preset several optimizations of the basic clausification algorithm. The *Theorem Prover for* $\mathcal{BS}$ implements the basic superposition calculus [14], which is used in the reduction algorithm. The design of this component is discussed in more detail in Section 12.3. Finally, the *Disjunctive Datalog Engine* is used for answering queries in the program obtained by the reduction. The design of this component is discussed in more detail in Section 12.4.

## 12.2 Ontology Clausification

The saturation of TBox and ABox clauses by basic superposition is a critical step of the reduction algorithm, because it can introduce an exponential blowup. Each clause

[3] `http://www.w3.org/TR/owl-semantics/`
[4] `http://www.w3.org/TR/owl-xmlsyntax/`
[5] `http://progete.stanford.edu/`
[6] `http://oiled.man.ac.uk/`

can potentially participate in an inference that generates many conclusions, so it is important to keep the size of the input clause set minimal. To achieve this, we present several optimizations of the clausification algorithm from Definition 5.3.1; they are also applicable to the clausification algorithm from Definition 8.2.2.

### 12.2.1 Reusing Replacement Predicates

A concept often occurs in many other concepts in $KB$, which allows us to reuse the replacement concepts:

**Definition 12.2.1.** *For $Q^D$ and $Q_D$ new concepts unique for $D$, the* optimized clausification *is obtained by modifying Definition 5.3.1 to use for $Q$ the concept as follows:*

$$Q = \begin{cases} Q^D & \text{if } \mathsf{pol}(C,p) = 1 \text{ and } C|_p = D \\ Q_D & \text{if } \mathsf{pol}(C,p) = -1 \text{ and } C|_p = D \end{cases}$$

For example, consider the knowledge base $KB$, consisting of axioms (12.1)–(12.3) shown on the left-hand side:

$$A \sqsubseteq \exists S.\exists R.D \quad \rightsquigarrow \quad \begin{array}{l} A \sqsubseteq \exists S.Q^{\exists R.D} \\ Q^{\exists R.D} \sqsubseteq \exists R.D \end{array} \tag{12.1}$$

$$B \sqsubseteq \exists T.\exists R.D \quad \rightsquigarrow \quad B \sqsubseteq \exists T.Q^{\exists R.D} \tag{12.2}$$

$$\exists U.\exists R.D \sqsubseteq C \quad \rightsquigarrow \quad \begin{array}{l} \exists U.Q_{\exists R.D} \sqsubseteq C \\ \exists R.D \sqsubseteq Q_{\exists R.D} \end{array} \tag{12.3}$$

The concept $\exists R.D$ occurs in $KB$ twice under positive, and once under negative polarity. Hence, we replace $\exists R.D$ in (12.1) and (12.2) with $Q^{\exists R.D}$, and in (12.3) with $Q_{\exists R.D}$, yielding the axioms shown in (12.1)–(12.3) on the right-hand side.

Note that $Q^{\exists R.D}$ should not be used to replace $\exists R.D$ in (12.3): the negation-normal form of (12.3) is $\forall U.\forall R.\neg D \sqcup C$, and it does not contain $\exists R.D$. Hence, the subconcept $\forall R.\neg D$ must be replaced by a concept different from $Q^{\exists R.D}$.

**Lemma 12.2.2.** *$KB$ and $\Xi(KB)$ are equisatisfiable, even if optimized structural transformation is applied to an $\mathcal{ALCHIQ}(\mathbf{D})$ knowledge base $KB$.*

*Proof.* The ($\Leftarrow$) direction is trivial, and the ($\Rightarrow$) direction can be proved by choosing the interpretation of $Q^D$ and $Q_D$ as done for decomposition in Subsection 5.4.1. □

### 12.2.2 Optional Positions

Certain nonliteral predicates need not be replaced with an atomic concept in the clausification process. Consider clausification of an axiom $A \sqsubseteq \exists R.B$: by Definition 5.3.1, $\mathsf{Cls}(\neg A \sqcup \exists R.B) = \{\neg A \sqcup Q_1,\ \neg Q_1 \sqcup \exists R.B\}$; that is, the concept $\exists R.B$ is replaced by $Q_1$. However, clausifying $\neg A \sqcup \exists R.B$ produces clauses of the form from Table 5.2 even without replacing the nonliteral subconcept $\exists R.B$. This allows for the following optimization:

**Definition 12.2.3.** *In a concept $C = D \sqcup \bigsqcup L_i$, where $D$ is a possibly complex concept and all $L_i$ are literal concepts, the positions of $D$ and $L_i$ are* optional for renaming.

**Lemma 12.2.4.** *Each clause in $\Xi(KB)$ is of a type from Table 5.2, even if $\mathsf{Def}(C)$ is computed by omitting in $\Lambda(C)$ the positions that are optional for renaming.*

*Proof.* By definition of $\pi$ from Table 3.1, clausifying a concept $C = D \sqcup \bigsqcup L_i$ produces clauses from Table 5.2. □

Renaming optional positions can sometimes be beneficial. Consider the knowledge base $KB$, consisting of axioms shown in (12.4)–(12.6) on the left-hand side:

$$A \sqsubseteq \exists R.C \quad \rightsquigarrow \quad \begin{array}{l} \neg A(x) \vee R(x, f(x)) \\ \neg A(x) \vee C(f(x)) \end{array} \tag{12.4}$$

$$B \sqsubseteq \exists R.C \quad \rightsquigarrow \quad \begin{array}{l} \neg B(x) \vee R(x, g(x)) \\ \neg B(x) \vee C(g(x)) \end{array} \tag{12.5}$$

$$\top \sqsubseteq \leq 1\, R \quad \rightsquigarrow \quad \neg R(x, y_1) \vee \neg R(x, y_2) \vee y_1 \approx y_2 \tag{12.6}$$

In $\neg A \sqcup \exists R.C$ and $\neg B \sqcup \exists R.C$, the positions of $\exists R.C$ are optional for renaming, so $KB$ can be clausified without introducing new predicates. This yields clauses shown in (12.4)–(12.6) on the right-hand side.

The concept $\exists R.C$ is now skolemized twice, yielding clauses $\neg A(x) \vee R(x, f(x))$ and $\neg B(x) \vee R(x, g(x))$ as candidates for an inference with (12.6). Additional axioms of the form $A_i \sqsubseteq \exists R.C$ would produce additional clauses of the form $\neg A_i(x) \vee R(x, f(x))$, which could participate in an inference with (12.6), thus significantly increasing the search space.

The search space can be reduced by renaming $\exists R.C$, although it occurs at optional positions in axioms. The axioms and the clauses obtained by applying the structural transformation to $KB$ are shown in (12.7)–(12.10):

$$A \sqsubseteq Q \quad \rightsquigarrow \quad \neg A(x) \vee Q(x) \tag{12.7}$$

$$B \sqsubseteq Q \quad \rightsquigarrow \quad \neg B(x) \vee Q(x) \tag{12.8}$$

$$Q \sqsubseteq \exists R.C \quad \rightsquigarrow \quad \begin{array}{l} \neg Q(x) \vee R(x, f(x)) \\ \neg Q(x) \vee C(g(x)) \end{array} \tag{12.9}$$

$$\top \sqsubseteq \leq 1\, R \quad \rightsquigarrow \quad \neg R(x, y_1) \vee \neg R(x, y_2) \vee y_1 \approx y_2 \tag{12.10}$$

This set contains one clause $\neg Q(x) \vee R(x, f(x))$, which can participate in an inference with (12.10); also, any additional axiom $A_i \sqsubseteq \exists R.C$ produces only a clause $\neg A_i(x) \vee Q(x)$. Note that renaming $\exists R.C$ pays off because the concept $\exists R.C$ occurs in $KB$ twice.

Hence, in Definition 5.3.1, $\Lambda(C)$ should be the set of all positions $p \neq \epsilon$ in $C$ such that (*i*) $C|_p$ is not a literal concept, and, for all positions $q$ below $p$, $C|_q$ is a literal concept; and (*ii*) $p$ is either not optional for renaming, or $C|_p$ occurs in $KB$ more than once.

### 12.2.3 Handling Functional Roles

Consider the knowledge base $KB$, containing the axioms shown in (12.11)–(12.13) on the left-hand side:

$$\top \sqsubseteq \leq 1\, R \quad \rightsquigarrow \quad \neg R(x, y_1) \vee \neg R(x, y_2) \vee y_1 \approx y_2 \tag{12.11}$$

$$A \sqsubseteq \exists R.C \quad \rightsquigarrow \quad \begin{array}{l} \neg A(x) \vee R(x, f(x)) \\ \neg A(x) \vee C(f(x)) \end{array} \tag{12.12}$$

$$B \sqsubseteq \exists R.D \quad \rightsquigarrow \quad \begin{array}{l} \neg B(x) \vee R(x, f(x)) \\ \neg B(x) \vee D(f(x)) \end{array} \tag{12.13}$$

The role $R$ is functional by (12.11), which means that an object in a model can have at most one $R$-successor. This allows using the same function symbol $f$ in skolemizing $\exists R.C$ and $\exists R.D$, yielding clauses (12.11)–(12.13) shown on the right-hand side.

The main benefit of such clausification is that resolving $\neg A(x) \vee R(x, f(x))$ and $\neg B(x) \vee R(x, f(x))$ with (12.11) produces a clause with a literal $f(x) \approx f(x)$; such a clause is a tautology (and can be deleted). Without reusing function symbols, clausifying (12.12) would produce $\neg B(x) \vee R(x, g(x))$, which, resolved with $\neg A(x) \vee R(x, f(x))$ and (12.11), produces a clause containing a literal $f(x) \approx g(x)$. Such a clause is not a tautology, and should be used in further inferences.

This optimization is also very important for the reduction to disjunctive datalog. Namely, for each function symbol $f$ occurring in $\Xi(KB)$, the program $\mathsf{DD}(KB)$ contains a new predicate $S_f$. By reducing the number of function symbols, we reduce the number of datalog predicates.

**Definition 12.2.5.** Clausification with optimized skolemization *is obtained from Definition 5.3.1 such that, if* $\top \sqsubseteq \leq 1\, R \in KB$*, then the existential quantifiers in* $\exists R.C$ *and* $\geq n\, R.C$ *are skolemized using a function symbol* $f_R$ *unique for* $R$*.*

**Lemma 12.2.6.** *$KB$ and $\Xi(KB)$ are equisatisfiable, even if clausification with optimized skolemization is used.*

*Proof.* If $R$ is functional, then an object $x$ is allowed to have at least one $R$-successor in a model $I$, which is represented by the functional term $f_R(x)$. □

### 12.2.4 Discussion

The techniques presented in this chapter are all geared towards reducing the number of clauses in $\Xi(KB)$. It is instructive to compare the effects that such optimizations would have in a tableau calculus such as [73].

Tableau calculi try to construct a model of a knowledge base by applying tableau expansion rules if some tableau constraint is not satisfied. Consider, for example, a knowledge base $KB$ consisting of the following axioms:

$$C(a) \tag{12.14}$$

$$R(a, b) \tag{12.15}$$

$$D(b) \tag{12.16}$$

$$C \sqsubseteq \exists R.D \tag{12.17}$$

The axiom (12.17), when applied to (12.14), yields a constraint $\exists R.D(a)$, which does not need to be further expanded: $a$ already has an $R$-successor $b$ that is a member of $D$, so there is no need to introduce a new $R$-successor for $a$.

The situation is quite different in resolution calculi. The clausification algorithm translates (12.17) into these two clauses:

$$\neg C(x) \vee R(x, f(x)) \tag{12.18}$$

$$\neg C(x) \vee D(f(x)) \tag{12.19}$$

In resolution, skolemization plays a role that is analogous to expanding existential concepts in tableau calculi: in clauses (12.18) and (12.19), the term $f(x)$ plays the role of the existentially implied $R$-successor for $x$. Thus, skolemization expands all existential quantifiers before the theorem proving process starts, regardless of whether it is actually necessary to do so in order to build a model.

Another important difference between resolution and tableau is that clauses obtained by skolemization are nonground; that is, they encode successor information for many individuals. For example, (12.18) and (12.19) encode successor information for any individual $x$ that is a member of $C$. When resolution inferences are applied to such nonground clauses, they act on the set of all those successors. In contrast, a tableau calculus expands each successor separately, and repeats an inference step for each successor separately.

The optimizations from Subsections 12.2.1, 12.2.2, and 12.2.3 can be understood as an attempt to reduce the number of existential individuals introduced in advance.

## 12.3 The Theorem Prover for $\mathcal{BS}$

As discussed in [116], implementing an efficient theorem prover is a very complex task. Apart from excellent software engineering skills, it also requires knowledge of numerous techniques in order to implement various modules of the system.

However, our goal is not to provide a general first-order theorem prover applicable to any first-order problem. Rather, we need a theorem prover that can handle only $\mathcal{ALCHIQ}$-closures. As shown in Table 5.2, such closures are of a rather limited form. Hence, general implementation techniques can be radically simplified, thus yielding a simpler, leaner, and more efficient implementation.

### 12.3.1 Inference Loop

The main design element of a resolution-based theorem prover is the *inference loop*, which is responsible for saturating a set of closures under the given set of inference and redundancy elimination rules. In KAON2, we use a simplified variant of the *DISCOUNT loop*, introduced first in the equational theorem prover E [133]. The pseudo-code of the inference loop is presented in Algorithm 12.1.

The inference loop is parameterized with a set of inference rules $\mathcal{C}$ (explained in more detail in Subsection 12.3.3) and a set of redundancy elimination rules $\mathcal{D}$ (explained in more detail in Subsection 12.3.4), and is given a set of closures $N$ to be saturated. To ensure that the saturation is fair—that is, that each nonredundant inference is eventually performed—, the closures are partitioned into two sets. The set $U$ is the set of *unprocessed* closures—that is, closures that have not taken part in any inference. The set $W$ is the set of *worked-off* closures—that is, closures for which all inferences with other members from $W$ have been performed.

In the beginning, $U$ contains the set of closures $N$ to be saturated and $W$ is empty. In each step of the saturation, a *given closure* $g$ is don't-care nondeterministically chosen and removed from $U$. We describe the algorithm for choosing the given closure, encapsulated in the function PickGivenClosure, in more detail in Subsection 12.3.7.

Next, redundancy elimination rules are applied sequentially to $g$, transforming it into a potentially simpler closure. We use a convention that, if a closure is made redundant by a simplification rule, then SimplifyClosure returns *null*. We discuss the redundancy elimination rules of KAON2 in Subsection 12.3.4.

If the simplified closure is not redundant and it is not equal to the empty closure, then *backward subsumption* is performed: using RetrieveSubsumedClosures, all closures from $W$ that are properly subsumed by $g$ are retrieved and removed from $W$. We describe the index structures used to optimize the retrieval of subsumed closures in Subsection 12.3.8.

Finally, all inferences between $g$ and the closures from $W$ are performed. All conclusions that are not syntactic tautologies are added to the set of unprocessed closures $U$, after which the entire process is repeated.

If the given closure is the empty closure, then $N$ is unsatisfiable. Otherwise, the process terminates if $U$ becomes empty; at this point, $W$ is the saturated set.

### 12.3.2 Representing $\mathcal{ALCHIQ}$-Closures

$\mathcal{ALCHIQ}$-closures of types 3, 4, 5, 6, and 8 have at most one variable, and any LPO is total on such terms and literals. This allows us to represent closures as totally ordered arrays without repeated literals. This has two main benefits. First, the maximal literal is the first literal, so the literal on which an inference should take place can be located in constant time. Second, conclusions between inferences can be computed using merge sort in time that is linear in the size of the premises.

Moreover, the restricted form of $\mathcal{ALCHIQ}$-closures allows us to use a polynomial subsumption checking algorithm. Namely, the general subsumption problem is known

**Algorithm 12.1** The Inference Loop of KAON2

**Input:**
$\mathcal{C}$: the set of inference rules
$\mathcal{D}$: the set of redundancy elimination rules
$N$: the set of closures to be saturated

**Output:**
the set of closures $W$ obtained by saturating $N$ by $\mathcal{C}$ and $\mathcal{D}$

**Local variables:**
$W$: the set of worked-off closures
$U$: the set of unprocessed closures

```
 1: W ⟵ ∅
 2: U ⟵ N
 3: while U ≠ ∅ do
 4:     g ⟵ PickGivenClosure(U)
 5:     U ⟵ U \ {g}
 6:     for each redundancy elimination rule r ∈ 𝒟 do
 7:         g ⟵ SimplifyClosure(g, r, W)
 8:     end for
 9:     if g ≠ null then
10:         if g = □ then
11:             return {□}                          ▷ N is unsatisfiable
12:         end if
13:         S ⟵ RetrieveSubsumedClosures(g, W)
14:         W ⟵ (W \ S) ∪ {g}
15:         T ⟵ ∅
16:         for each inference rule r ∈ 𝒞 do
17:             T ⟵ T ∪ ComputeConclusions(r, g, W)
18:         end for
19:         for each conclusion t ∈ T do
20:             if IsSyntacticTautology(t) = false then
21:                 U ⟵ U ∪ {t}
22:             end if
23:         end for
24:     end if
25: end while
26: return W                                        ▷ W is the saturated set
```

to be NP-complete [52], and experience has shown that theorem provers spend up to 90 percent of running time in subsumption checking. Therefore, improving the efficiency of subsumption is expected to have a significant effect on the performance.

The subsumption checking procedure for closures of types 3, 4, 5, 6, and 8 is shown in Algorithm 12.2. To determine if $C$ subsumes $D$, the algorithm essentially scans the literals of $D$ (lines 1–14), trying to match then with the first literal of $C$ (line 2). If a substitution $\sigma$ exists such that $C_1\sigma = D_i$, because $C_1$ contains all variables of $C$, the substitution $\sigma$ binds all variables in $C$. Hence, in lines 5–8 the algorithm simply checks whether, for each literal $C_i$, one can find a literal $D_j$ such that $C_i\sigma = D_j$ and that each marked position of $C_i\sigma$ can be overlaid in a substitution position in $D_j$. Since the closures are totally ordered, the literals can be matched sequentially. To estimate the complexity, note that the inner loop (lines 1–14) is executed at most $|D|$ times, whereas the inner loop (lines 5–8) is executed at most $|C| + |D|$ times. Hence, the algorithm runs in time $\mathcal{O}(|C| \cdot (|C| + |D|))$, which is obviously polynomial.

Additionally, we store with each closure a flag specifying its type, as this enables further optimizations. For example, a closure of type 3 or 4 cannot subsume a closure of any other type. Similarly, closures of types 1, 2, or 7 can subsume only closures of the same type. Hence, a quick check whether closure types are compatible can be used to quickly identify subsumption tests that are bound to fail.

### 12.3.3 Inference Rules

The theorem prover of KAON2 encodes the literals with a predicate other than equality as $\mathcal{E}$-terms (see Section 2.5), allowing a more or less straightforward implementation of the inferences rules of the $\mathcal{BS}$ calculus. The implementation can be somewhat simplified by observing that closures of types 1, 2, and 7 can only participate in hyper-resolution inferences, in which unification can be performed in constant time. Hence, the theorem prover handles closures of types 1, 2, and 7 by a separate hyperresolution inference rule, whereas inferences among other closures are handled using positive and negative superposition.

Furthermore, negative superposition inferences can result in closures containing literals of the form $\mathsf{T} \not\approx \mathsf{T}$ and $x \not\approx x$. For such closures, reflexivity resolution produces a closure that always subsumes the superposition conclusion, and is therefore applied eagerly to each conclusion.

### 12.3.4 Redundancy Elimination Rules

The simplification rules are applied to the given closure in the order as explained in the following paragraphs. It may come as a surprise that KAON2 does not implement so-called rewriting simplification inferences. Namely, given a unit closure $s \approx t$ and a closure $A \vee C$, under certain conditions, $A \vee C$ can be replaced with $A[t\sigma]_p \vee C$, where $\sigma$ is a substitution such that $A|_p = s\sigma$. In general first-order theorem provers, such as E [133] or Vampire [119], such inferences are essential for solving hard problems. However, most DL problems do not contain unit equalities at all. Equalities occur only

**Algorithm 12.2** Closure Subsumption Algorithm in KAON2

**Input:**
$C$: the potential subsumer
$D$: the potential subsuming closure
**Output:**
a substitution $\sigma$ such that $C\sigma \subseteq D$
**Note:**
$C_i$ is the $i$-th literal of a closure $C$, and $|C|$ is the number of literals in $C$.

```
1: for i = 1...|D| do
2:     σ = MATCH(C_1, D_i)
3:     if σ ≠ null then
4:         j ⟵ 0
5:         for k = i...|D| and j < |C| do
6:             if C_jσ = D_k and MARKERSCOMPATIBLE(C_jσ, D_k) then
7:                 j ⟵ j + 1
8:             end if
9:         end for
10:        if j = |C| then
11:            return σ
12:        end if
13:    end if
14: end for
15: return null                                   ▷ No subsumption

16: function MATCH(s,t)
17:    if s is a variable x then
18:        return {x ↦ t}
19:    else if s = f(s_1,...,s_n) and t = f(t_1,...,t_n) then
20:        σ ⟵ {}
21:        for i = 1...n do
22:            θ ⟵ MATCH(s_iσ, t_i)
23:            if θ = null then
24:                return null
25:            else
26:                σ ⟵ σθ
27:            end if
28:        end for
29:        return σ
30:    else
31:        return null
32:    end if
33: end function

34: function MARKERSCOMPATIBLE(s,t)
35:    true if t|_p is at substitution position whenever s|_p is marked, for each position p
36: end function
```

in $\mathcal{ALCHIQ}$-closures of type 5, 6, or 8, where they are usually accompanied by at least one literal with a predicate other than $\approx$. Hence, rewriting is not applicable to the vast majority of DL problems, and we therefore do not include it into our implementation.

**Semantic Tautology Deletion.** As mentioned in Section 2.4, checking if a closure is a tautology requires theorem proving, so simple syntactic approximate checks are used in practice. However, a feasible semantic check was presented in [133]: a closure is a tautology if it contains a disjunction of the form $s_1 \not\approx t_1 \vee \ldots \vee s_n \not\approx t_n \vee s \approx t$ such that $s_1\sigma \approx t_1\sigma, \ldots, s_n\sigma \approx t_n\sigma \models s\sigma \approx t\sigma$, where $\sigma$ is a substitution mapping all variables in the closure to distinct new constants.

Since deciding entailment in a ground equality theory is decidable, the check from the previous paragraph is decidable, but implementing it may introduce a significant amount of overhead. However, the syntactic structure of $\mathcal{ALCHIQ}$-closures makes the check easy to implement in certain situations. Namely, it is easy to see that the conditions of semantic tautology deletion are satisfied for any closure of type 5 or 6 that contains a disjunction of the following form:

$$f_1(t) \not\approx f_2(t) \vee f_2(t) \not\approx f_3(t) \vee \ldots \vee f_{n-1}(t) \not\approx f_n(t) \vee f_1(t) \approx f_n(t)$$

Namely, for any substitution $\sigma$, it is easy to see that the following condition holds:

$$f_1(t\sigma) \approx f_2(t\sigma) \wedge f_2(t\sigma) \approx f_3(t\sigma) \wedge \ldots \wedge f_{n-1}(t\sigma) \approx f_n(t\sigma) \models f_1(t\sigma) \approx f_n(t\sigma)$$

Hence, semantic tautology deletion can be applied to a closure $C$ of type 5 or 6 as follows. We first construct an unoriented graph $G_C$ whose nodes are function symbols from $C$, and where nodes $f_i$ and $f_j$ are connected if $C$ contains a literal $f_i(t) \not\approx f_j(t)$. Next, we compute the transitive closure $G_C^+$ of $G_C$. Finally, we check if $C$ contains a literal $f_i(t) \approx f_j(t)$ such that $f_i$ and $f_j$ are connected by an edge in $G_C^+$. This check can be performed in time that is polynomial in $|C|$, and, if it succeeds, $C$ is deleted.

**Forward Subsumption.** A closure $C$ is deleted if the set of worked-off closures $W$ contains a closure $D$ that subsumes $C$. We discuss the index structures used to optimize subsumption checking in more detail in Subsection 12.3.8.

**Contextual Literal Cutting.** This subsumption-based simplification rule [134] allows eliminating certain literals from a closure. Namely, a closure $C \vee L$ can be replaced with $C$, provided that $C \vee \overline{L}$ is subsumed in the set of worked-off closures $W$. Hence, contextual literal cutting can be implemented by complementing each literal, and then checking whether the resulting closure is subsumed in $W$.

The correctness of the rule can intuitively be explained as follows: let $D$ be a closure from $W$ that subsumes $C \vee \overline{L}$ through a substitution $\sigma$—that is, $D\sigma \subseteq C \vee \overline{L}$. Furthermore, because $D$ is in $W$, we have $W \models D\sigma \vee \overline{L}$. But now we can resolve $C \vee L$ with $D\sigma \vee \overline{L}$; because $D\sigma \in C$, we obtain the closure $C$, which obviously subsumes $C \vee L$. Hence, contextual literal cutting can be understood as a macro, combining the effects of the mentioned inferences.

### 12.3.5 Optimizing Number Restrictions

It is not difficult to see that resolution-based techniques are not well suited for reasoning with number restrictions. Namely, a concept of the form $\leq n\, R.C$ is translated into a closure of type 7, containing $\frac{(n+1)n}{2}$ literals of the form $y_i \approx y_j$. If such a closure is hyperresolved with $m$ $R$-generators of type 3, each hyperresolution produces closures containing many literals of the form $[f_i(x)] \approx [f_j(x)]$. Furthermore, such a literal is produced by each inference with a pair of side premises containing $f_i$ and $f_j$, thus resulting in many similar closures. Because each literal $f_i(x) \approx f_j(x)$ is a potential source of a superposition inference, number restrictions dramatically increase the number of superposition inferences in a saturation.

The number of superposition inferences can be somewhat reduced by applying decomposition and replacing literals of the form $[f_i(t)] \approx [f_j(t)]$ with $Q_{f_i,f_j}([t])$. Consider the closures (12.20)–(12.21), obtained by resolving $C_i(x) \vee R(a, f_i(x))$, $1 \leq i \leq 4$, with $\neg R(x, y_1) \vee \neg R(x, y_2) \vee \neg R(x, y_3) \vee y_1 \approx y_2 \vee y_2 \approx y_3 \vee y_1 \approx y_3$:

$$C \vee [f_1(x)] \approx [f_2(x)] \vee [f_1(x)] \approx [f_3(x)] \vee [f_2(x)] \approx [f_3(x)] \tag{12.20}$$

$$D \vee [f_1(x)] \approx [f_2(x)] \vee [f_1(x)] \approx [f_4(x)] \vee [f_2(x)] \approx [f_4(x)] \tag{12.21}$$

Using decomposition, (12.20)–(12.21) can be transformed into (12.22)–(12.28):

$$C \vee Q_{f_1,f_2}(x) \vee Q_{f_1,f_3}(x) \vee Q_{f_2,f_3}(x) \tag{12.22}$$

$$D \vee Q_{f_1,f_2}(x) \vee Q_{f_1,f_4}(x) \vee Q_{f_2,f_4}(x) \tag{12.23}$$

$$\neg Q_{f_1,f_2}(x) \vee [f_1(x)] \approx [f_2(x)] \tag{12.24}$$

$$\neg Q_{f_1,f_3}(x) \vee [f_1(x)] \approx [f_3(x)] \tag{12.25}$$

$$\neg Q_{f_1,f_4}(x) \vee [f_1(x)] \approx [f_4(x)] \tag{12.26}$$

$$\neg Q_{f_2,f_3}(x) \vee [f_2(x)] \approx [f_3(x)] \tag{12.27}$$

$$\neg Q_{f_2,f_4}(x) \vee [f_2(x)] \approx [f_4(x)] \tag{12.28}$$

Now there is exactly one closure containing a literal of the form $[f_i(x)] \approx [f_j(x)]$, so superposition from such a literal is performed only once, and not for each closure where $[f_i(x)] \approx [f_j(x)]$ occurs.

A disadvantage of this transformation is that it does not allow us to apply semantic tautology deletion (see Subsection 12.3.4), since literals $f_i(x) \approx f_j(x)$ occur only in isolation. This can be remedied as follows: for each pair of function symbols $f_i$ and $f_j$, we additionally introduce a closure $Q_{f_i,f_j}(x) \vee [f_i(x)] \not\approx [f_j(x)]$, which makes $Q_{f_i,f_j}(x)$ equivalent to $[f_i(x)] \approx [f_j(x)]$. Now semantic tautology deletion can be applied as described previously with the difference that, instead of $[f_i(x)] \approx [f_j(x)]$, the literals $Q_{f_i,f_j}([t])$ are used in building the graph $G_C$.

### 12.3.6 Tuning the Calculus Parameters

The main role of the parameters of $\mathcal{BS}_{DL}$ is to guarantee termination of the calculus. However, Definition 5.3.3 provides some room for fine-tuning the parameters.

More concretely, we can choose the precedence of predicate symbols $>$ based on the frequency of occurrences of symbols in a knowledge base. Practical experiments have shown that in many cases, it is beneficial to use a precedence based on decreasing frequency of symbols—that is, a precedence where the most-frequent predicate is the smallest in the precedence.

Furthermore, we can tweak the selection function. It is known that selecting and hyperresolving multiple negative literals in practice often leads to better performance. Intuitively, such inferences are efficient because they do not derive many intermediary consequences. In the case of $\mathcal{BS}_{DL}$, we can use a selection function that selects negative literals containing the deepest terms in a closure. Lemma 5.3.6 still holds in such a case, because Lemma 5.3.5 is still satisfied.

### 12.3.7 Choosing the Given Closure

General theorem provers are geared towards solving hard problems in first-order logic. However, first-order logic is semidecidable: given a finite amount of time, we can hope only to detect unsatisfiability. For satisfiable problems, no guarantee of finding a solution exists. Therefore, most existing theorem provers are geared towards detecting unsatisfiability.

It is widely believed that many hard first-order problems have relatively short proofs. Hence, the performance of first-order reasoning is expected to depend a lot on choosing a good proof strategy, which is mainly determined by the algorithm for choosing the given closure (line 4 of Algorithm 12.1). Therefore, numerous different heuristics for choosing the given closure have been explored.

Unlike most general first-order provers, the theorem prover in KAON2 is applied mostly to satisfiable problems. Namely, ontologies are typically satisfiable (if this is not so, they usually contain an error). Furthermore, in computing a reduction to disjunctive datalog, the saturation of the TBox axioms is computed. Finally, our experience and that of other authors of DL reasoning systems shows that most problems encountered while computing a subsumption hierarchy are satisfiable. Hence, the strategy for choosing the given closure is less important in our case, because the theorem prover in most cases performs all inferences anyway.

Therefore, we adopt a simple strategy of always choosing a closure from $U$ that has the smallest number of literals, and of resolving ties in an arbitrary manner. Intuitively, a short closure is more likely to subsume a closure in $W$.

### 12.3.8 Indexing Terms and Closures

As discussed in [116], a critical aspect of an implementation of a first-order theorem prover is to provide appropriate *clause indexes*—auxiliary data structures used to efficiently identify closures used at particular steps in the saturation. *Unification indexes* are used to retrieve potential candidate closures that can participate in an inference, whereas *subsumption indexes* are used to retrieve closures that subsume or that are subsumed by a certain closure.

Many different indexing techniques were developed; for an extensive overview, please refer to [79]. A prominent indexing technique is *partially adaptive code trees* [117], pioneered in the theorem prover Vampire [119]. Whereas this technique is known to give excellent performance, the restricted structure of $\mathcal{ALCHIQ}$-closures allows us to employ simpler, but still efficient indexing techniques.

**Unification Indexes.** The problem of unification indexing can be formalized as follows: given a query term $s$, a unification index identifies the subset $M$ of a set of terms $N$ such that each term $t \in M$ unifies with $s$. In practice, one usually also associates with each term $t$ a closure in which $t$ occurs.

Consider now the type of $\mathcal{BS}_{DL}$ inferences applicable to $\mathcal{ALCHIQ}$-closures. The candidates for (hyper)resolution with a closure of type 1, 2, or 7 are closures of types 3, 4, or 8, and their suitability is determined uniquely by the role symbol. Hence, indexing can be simply performed by hashing the closures on the role symbol.

More complex are superposition inferences between closures of types 5 and 6, superposition into generator closures, and superposition into unary literals of closures of type 8. These closures can be indexed using a simplified variant of the well-known *path indexing technique* [140, 113]. Namely, inferences with these closures take place only on unary $\mathcal{E}$-terms, which contain at most one variable. This allows for the following simple indexing strategy. Each unary $\mathcal{E}$-term $f_1(f_2(\ldots))$ can be represented as a string of function symbols $f_1.f_2\ldots$, and by encoding variables using a special symbol $*$. To build an index over a set of terms, we use the well-known trie data structure over the strings (for more information on the trie data structures, please refer to [137]). Given a term $s$, locating all terms from the trie can be performed by traversing the trie using the string corresponding to $s$. The trie traversal algorithm is standard, with a minor difference that the symbol $*$ can be matched with zero or more arbitrary symbols.

Moreover, observe that, for closures as described in the previous paragraph, a superposition inference can be performed either into the outermost position $p = \epsilon$ (for example, on a literal $A(f(x))$), into the position $p = 1$ (for example, at position of $f(x)$ in a literal $A(f(x))$), or into the position $p = 2$ (for example, at position of $f(x)$ in a literal $R(x, f(x))$); at all other positions, the terms are always marked. Hence, the closures that are possible targets of superposition inferences are indexed using three indexes $I_\epsilon$, $I_1$, and $I_2$. Furthermore, superposition can be performed only from the outermost position of a positive equality, so the closures that are possible sources of superposition inferences are indexed using a single index $F_\epsilon$.

A closure $C$ is then indexed as follows. First, the literal $L$ eligible for inference is determined. Then, if $L$ is a positive literal of the form $s \approx t$, the term $s$ is added to $F_\epsilon$. Finally, all positions $p \in \{\epsilon, 1, 2\}$ are examined. If $L|_p$ is not at substitution position, and either $p \neq \epsilon$ or $L$ is not of the form $A \approx \mathsf{T}$, then the term $L|_p$ is added to $I_p$. The case of $p = \epsilon$ and $L$ of the form $A \approx \mathsf{T}$ is treated differently because performing superposition into such $A$ always results in a tautology $\mathsf{T} \approx \mathsf{T}$; such inferences are prevented by simply not entering $A$ into $I_\epsilon$.

**Subsumption Indexes.** Subsumption indexes are used for two related problems. *Forward subsumption* is the problem of determining whether a set of closures $N$ contains a closure that subsumes some closure $C$, whereas *backward subsumption* is the problem of determining the subset $M$ of a set of closures $N$ such that $C$ subsumes each closure in $M$.

The main problem in subsumption indexing is that, if $C\sigma \subseteq D$ for some substitution $\sigma$, the literals in $D$ matched to the literals of $C\sigma$ need not be consecutive. For example, let $C = A_2(x) \vee A_5(x)$ and let $D = A_1(x) \vee A_2(x) \vee A_3(x) \vee A_4(x) \vee A_5(x) \vee A_6(x)$. It is clear that $C$ subsumes $D$; however, the closure $D$ contains the disjunction $A_3(x) \vee A_4(x)$ between literals $A_2(x)$ and $A_5(x)$. Because $C$ can be matched in general to an arbitrary subset of $D$ and there are exponentially many such subsets, it is difficult to design an efficient indexing structure.

In KAON2, we apply an approximate technique, inspired by *feature vector indexing* from [134]. Our algorithm works as follows. To index a closure $C$, we build a string $C_1.C_2 \ldots C_n$ of predicate names occurring in $C$, order it by the LPO precedence $>$, and insert it into a trie.

For forward subsumption, given a closure $D$, we build a string $*.D_1.*.D_2.*\ldots*.D_n.*$ of predicate names occurring in $D$, order it by the LPO precedence $>$, and use it to search the trie. As a result, we obtain a set of closures containing the predicates of $D$. Hence, the retrieved closures are potential subsumers of $D$. For each such closure, we check subsumption using Algorithm 12.2.

For backward subsumption, given a closure $D$, we build a string $D_1.D_2 \ldots D_n$ of predicate names occurring in $D$ and order it by the LPO precedence $>$. We then search the trie using this string, but we treat each trie entry as $*.C_1.*.C_2.*\ldots*.C_n.*$. This again gives us the set of potential subsumption candidates, which is then filtered using Algorithm 12.2.

## 12.4 Disjunctive Datalog Engine

Several disjunctive datalog engines, such as DLV [41] or Smodels [141], were implemented and successfully applied to practical problems. However, we decided to implement a new datalog engine for KAON2 because of the following three reasons.

First, interfacing with existing engines is currently difficult. Available systems were implemented in different programming languages. Furthermore, the primary mode of interaction with them is through command line. Efforts to enable access to existing systems by means of an established API are currently in progress, but such an API has not been available at the time of writing. Hence, implementing our own system seemed preferable to having to struggle with low-level implementation issues.

Second, available disjunctive datalog engines are designed to provide practical support for nonmonotonic reasoning under stable models. In contrast, disjunctive programs obtained by our algorithms are positive, so they do not require nonmonotonic reasoning. Hence, it seemed reasonable to expect that a more specialized implementation will provide better performance.

Third, nonground query answering in existing systems is typically solved by reducing the problem to ground query answering, and then checking the correctness of each answer using model building. We wanted to investigate whether nonground query answering can be better realized using direct methods, without grounding.

In the rest of this section we describe the important aspects of the implementation of our disjunctive datalog engine.

### 12.4.1 Magic Sets

After eliminating the equality predicate using the algorithm from Section 7.5, the magic sets transformation, presented in [34], is applied to $P \cup P_{\approx}$. A detailed presentation of the transformation is beyond the scope of this subsection; we just explain the intuition behind this technique.

Magic sets were motivated by a significant advantage of top-down over bottom-up query evaluation techniques. Namely, in a top-down algorithm, the query guides the computation. Consider the following program $P$:

$$T(x, y) \leftarrow R(x, y) \tag{12.29}$$

$$T(x, z) \leftarrow R(x, y), T(y, z) \tag{12.30}$$

A bottom-up algorithm for answering a query $T(a, x)$ in $P$ first computes all facts that follow from $P$, and then selects the tuples from $T$ containing $a$ in the first position. This is inefficient, because the algorithm computes entire transitive closure of $R$.

However, starting from the query, a top-down technique first retrieves only those tuples from $R$ that contain $a$ at the first position, and, for each such tuple, it recursively invokes $T(y, z)$. Hence, the bindings for $y$, determined by querying $R(a, y)$, guide the query answering process; this is also called *binding propagation*. Hence, a top-down engine does not compute the entire relation $T$, but only the part receiving the bindings from the query and the data.

Magic sets [18] simulate top-down binding propagation in bottom-up query evaluation. A program $P$ is modified by introducing *magic predicates*, whose role is to keep track of the bindings that would be propagated during top-down evaluation of a query $Q$ in $P$. Magic predicates are then used to constrain the rules of $P$ such that evaluating them performs only those inferences that are relevant to the query. Note that the magic program depends on the query $Q$; that is, for different queries, the magic program must be computed from scratch.

To the best of our knowledge, the first generalization of the magic sets techniques to nondisjunctive programs was presented in [55]. Apart from the theoretical results, the author provides empirical evidence for the benefits of the technique. The technique was further refined in [34]. Because of the simpler structure of the transformed programs, we decided to use the latter variant in KAON2.

In the presence of equality, magic set transformation becomes less effective. Consider, for example, the following equality-free program $P$:

$$C(x) \leftarrow D(x,y), E(y,z) \tag{12.31}$$

$$R(x,y) \leftarrow S(x,y) \tag{12.32}$$

To answer the query $C(x)$ in $P$, only the extensions of $C$, $D$, and $E$ are relevant; the extensions of $R$ and $S$ need not be considered. Moreover, magic sets are very effective at identifying the relevant parts of $D$ and $E$. Now let $P'$ be the program $P$ extended with (12.33), stating that $R$ is functional:

$$y_1 \approx y_2 \leftarrow R(x,y_1), R(x,y_2) \tag{12.33}$$

Answering the query $C(x)$ in $P'$ is now more involved because (12.33) can potentially make any two objects equal. If $C(a)$ is an answer in $P$, then in $P'$ the rule (12.33) should be invoked to check whether any other object is equal to $a$. Thus, the extensions of $R$ and $S$ are now relevant for the query.

### 12.4.2 Bottom-Up Saturation

After applying the magic sets, the query $Q$ is answered in $P \cup P_{\approx}$ using the algorithm from Definition 7.6.1. To avoid redundant computation of facts derived by rules, we apply the *general seminaïve strategy* [112]. A detailed presentation of the algorithm is beyond the scope of this subsection; we just explain the intuition behind it.

The algorithm is an optimization of the bottom-up saturation, developed to prevent recomputing the same conclusions. Consider a datalog program $P$ containing only the following rule:

$$C(y) \leftarrow R(x,y), C(x) \tag{12.34}$$

Moreover, let $F_0$ contain the facts $C(a_0)$ and $R(a_{i-1}, a_i)$ for $0 < i \leq n$. A query $C(x)$ can be answered in $P$ and $F_0$ using bottom-up saturation: we compute sets of facts $F_0, F_1, \ldots, F_m$, where each $F_i$ is obtained by applying the rules of $P$ to the facts in $F_{i-1}$. The saturation terminates when no new facts are derived—that is, when $F_m = F_{m-1}$. In our example, the saturation yields $F_i = F_{i-1} \cup \{C(a_i)\}$. Note that $F_{i-1} \subseteq F_i$; hence, the rules of $P$ that are applicable to facts in $F_k$ for $k < i$, are also applicable to $F_i$. Hence, each iteration computes only one additional fact, but to do so, it also recomputes all facts from $F_{i-1} \setminus F_0$.

Recomputing facts might be avoided by evaluating, instead of $P$, a sequence of programs $P_0, P_1, \ldots, P_m$, where each $P_i$ contains the following rule:

$$\Delta C_i(y) \leftarrow R(x,y), \Delta C_{i-1}(x)$$

The iteration is initialized by asserting $\Delta C_0(\alpha)$ for each $C(\alpha) \in F_0$, and the result is obtained as $C = \bigcup \Delta C_i$. By evaluating each $P_i$, only facts derived by $P_{i-1}$ are used, and no fact is computed twice.

The seminaïve evaluation algorithm uses this basic idea, while solving several practical issues, such as minimizing the number of extensions $\Delta C_i$ used in the computation. Furthermore, it can be combined with our resolution approach for answering queries in a disjunctive program from Section 7.6.

# Chapter 13

# Performance Evaluation

In this chapter we present a comparison of the performance of KAON2 with that of existing tableau DL reasoning systems, and thus obtain an insight into the practical applicability of our algorithms.

While conducting the experiments, we observed that it is very difficult, if not impossible, to separate the reasoning method from its implementation. Reasoning algorithms in general are quite complex, and to implement them efficiently, a lot of time should be invested in low-level implementation details. Overheads in maintaining various data structures or memory management can easily dominate the reasoning time. The choice of the implementation language also matters a lot, as its limitations may easily become critical in dealing with large data sets.

Therefore, the results we present in this chapter should not be taken as a definitive measure of practicability of either algorithm. However, they do show that deductive database techniques can significantly improve the performance of reasoning. This seems to hold even if the TBox is complex, as long as it is relatively small. For TBox reasoning, our results reflect a mixed picture: on certain ontologies the resolution algorithms perform relatively well, whereas, on other ontologies, they are slower than the tableau algorithms. This suggests that resolution-based techniques might be interesting in practice, but that further research is needed to match the robustness of tableau algorithms on TBox reasoning problems.

## 13.1 Test Setting

We compared KAON2 with RACER and Pellet. We did not consider other DL reasoners because they follow different design choices: FaCT[1] [67], FaCT++,[2] and DLP[3] do not support ABoxes; LOOM [91] is incomplete; and CLASSIC [22] supports only a very inexpressive logic.

[1] `http://www.cs.man.ac.uk/~horrocks/FaCT/`

[2] `http://owl.man.ac.uk/factplusplus/`

[3] `http://www.bell-labs.com/user/pfps/dlp/`

RACER[4] [59] was developed at the Concordia University and the Hamburg University of Technology, and is written in Common Lisp. We used the version 1.8.2, to which we connected using the JRacer library. RACER provides an optimized reasoning mode (so-called nRQL mode 1) that provides significant performance improvements, but which is complete only for certain types of knowledge bases. At the time of writing, RACER did not automatically recognize whether the optimized mode is applicable to a particular knowledge base, so we used RACER in the mode that guarantees completeness (so-called nRQL mode 3). Namely, determining whether optimizations are applicable is a form of reasoning that, we believe, should be taken into account in a fair comparison.

Pellet[5] [105] was developed at the University of Maryland, and it is the first system that fully supports OWL-DL, taking into account all the nuances of the specification. It is implemented in Java, and is freely available with the source code. We used the version 1.3 beta.

We asked the authors of each tool for an appropriate sequence of API calls for running tests. For each reasoning task, we started a fresh instance of the reasoner and loaded the test knowledge base. Then, we measured the time required to execute the task. We made sure that all systems return the same answers.

Many optimizations of tableau algorithms involve caching computation results, so the performance of query answering should increase with each subsequent query. Furthermore, both RACER and Pellet check ABox consistency before answering the first query, which typically takes much longer than computing query results. Hence, starting a new instance of the reasoner for each query might seem unfair. We justify our approach as follows.

First, the effectiveness of caching depends on the type of application: if an ABox changes frequently, caching is not very useful. Second, usefulness of caches also depends on the degree of similarity between queries. Third, we did not yet consider caching for KAON2; however, materialized views were extensively studied in deductive databases, and were applied in [148] to ontology reasoning. Finally, KAON2 does not perform a separate ABox consistency test because ABox inconsistency is discovered automatically during query evaluation. Hence, we decided to measure only the performance of the actual reasoning algorithm, and to leave a study of possible materialization and caching strategies for future work. Since ABox consistency check is a significant source of overhead for tableau systems, we measured the time required to execute it separately. Hence, in our tables, we distinguish one-time *setup* time (S) and query processing time (Q) for Pellet and Racer. The time for computing the datalog program in KAON2 was not significant, so we include it into the query processing time.

All tests were performed on a laptop computer with a 2 GHz Intel processor, 1 GB of RAM, running Windows XP Service Pack 2. For Java-based tools, we used Sun's Java 1.5.0 Update 5. The virtual memory of the Java virtual machine was limited to

[4] `http://www.racer-systems.com/`

[5] `http://www.mindswap.org/2003/pellet/index.shtml`

Table 13.1: Statistics of Test Ontologies

| *KB* | $C \sqsubseteq D$ | $C \equiv D$ | $C \sqcap D \sqsubseteq \bot$ | functional | domain | range | RBox | $C(a)$ | $R(a,b)$ |
|---|---|---|---|---|---|---|---|---|---|
| vicodi_0 | | | | | | | | 16942 | 36711 |
| vicodi_1 | | | | | | | | 33884 | 73422 |
| vicodi_2 | 193 | 0 | 0 | 0 | 10 | 10 | 10 | 50826 | 110133 |
| vicodi_3 | | | | | | | | 67768 | 146844 |
| vicodi_4 | | | | | | | | 84710 | 183555 |
| semintec_0 | | | | | | | | 17941 | 47248 |
| semintec_1 | | | | | | | | 35882 | 94496 |
| semintec_2 | 55 | 0 | 113 | 16 | 16 | 16 | 6 | 53823 | 141744 |
| semintec_3 | | | | | | | | 71764 | 188992 |
| semintec_4 | | | | | | | | 89705 | 236240 |
| lubm_1 | | | | | | | | 18128 | 49336 |
| lubm_2 | 36 | 6 | 0 | 0 | 25 | 18 | 9 | 40508 | 113463 |
| lubm_3 | | | | | | | | 58897 | 166682 |
| lubm_4 | | | | | | | | 83200 | 236514 |
| wine_0 | | | | | | | | 247 | 246 |
| wine_1 | | | | | | | | 741 | 738 |
| wine_2 | | | | | | | | 1235 | 1230 |
| wine_3 | | | | | | | | 1729 | 1722 |
| wine_4 | | | | | | | | 2223 | 2214 |
| wine_5 | 126 | 61 | 1 | 6 | 6 | 9 | 9 | 2717 | 2706 |
| wine_6 | | | | | | | | 5187 | 5166 |
| wine_7 | | | | | | | | 10127 | 10086 |
| wine_8 | | | | | | | | 20007 | 19926 |
| wine_9 | | | | | | | | 39767 | 39606 |
| wine_10 | | | | | | | | 79287 | 78966 |
| dolce | 203 | 27 | 42 | 2 | 253 | 253 | 522 | 0 | 0 |
| galen | 3237 | 699 | 0 | 133 | 0 | 0 | 287 | 0 | 0 |

800 MB, and each reasoning task was allowed to run for at most 5 minutes. Tests that ran either out of memory or out of time are denoted in tables with a value of 10000.

## 13.2 Test Ontologies

We based our tests on ontologies available in the Semantic Web community. To obtain sufficiently large test ontologies, we used ABox replication—duplication of ABox axioms with appropriate renaming of individuals. The information about the structure of ontologies we used is summarized in Table 13.1. All test ontologies are available on the KAON2 Web site.[6]

[6] `http://kaon2.semanticweb.org/download/test_ontologies.zip`

An ontology about European history was manually created in the EU-funded VICODI project.[7] The TBox is relatively small and simple: it consists of role and concept inclusion axioms, and domain and range specifications; it does not contain disjunctions, existential quantification, or number restrictions. However, the ABox is relatively large, with many interconnected instances. With vicodi_0, we denote the ontology from the project, and with vicodi_$n$ the one obtained by replicating $n$ times the ABox of vicodi_0.

An ontology about financial services was created in the SEMINTEC project[8] at the University of Poznan. Like VICODI, this ontology is relatively simple: it does not use existential quantifiers or disjunctions; it does, however, contain functionality assertions and disjointness. With semintec_0, we denote the ontology from the project, and with semintec_$n$ the one obtained by replicating $n$ times the ABox of semintec_0.

Lehigh University Benchmark (LUBM)[9] was developed by [58] as a benchmark for testing performance of ontology management and reasoning systems. The ontology describes organizational structure of universities, and is relatively simple: it does not use disjunctions or number restrictions, but it does use existential quantifiers, so it is in Horn-$\mathcal{ALCHI}$ fragment. Each lubm_$n$ is generated automatically by specifying the number $n$ of universities.

The Wine[10] ontology contains a classification of wines. It uses nominals, which our algorithms cannot handle. Therefore, we apply a sound but an incomplete approximation: we replace each enumerated concept $\{i_1, \ldots, i_n\}$ with a new concept $O$, and add assertions $O(i_k)$. The resulting ontology is relatively complex: it contains functionality axioms, disjunctions, and existential quantifiers. With wine_0, we denote the original ontology, and with wine_$n$ the one obtained by replicating $2^n$ times the ABox of wine_0.

This approximation of nominals is incomplete for query answering: for completeness one should further add a clause $\neg O(x) \vee x \approx i_1 \vee \ldots \vee x \approx i_n$. Furthermore, Pellet fully supports nominals, so one may question whether the Wine ontology is suitable for our tests. Unfortunately, in our search for test data, we could easily find ontologies with a complex TBox but without an ABox, or ontologies with an ABox and only a simple TBox, or a TBox with nominals. The (approximated) Wine ontology was the best ontology we found that contained both a nontrivial TBox and an ABox. We also used this approximated ontology in tests with Pellet, in order to ensure that all systems are dealing with the same problem.

DOLCE[11] is a foundational ontology developed at the Laboratory for Applied Ontology of Italian National Research Council. It is very complex, and no reasoner currently available can handle it. Therefore, the ontology has been factored into several

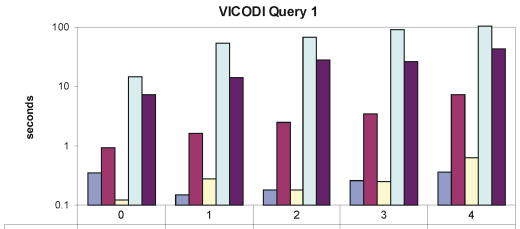
[7] http://www.vicodi.org/

[8] http://www.cs.put.poznan.pl/alawrynowicz/semintec.htm

[9] http://swat.cse.lehigh.edu/projects/lubm/index.htm

[10] http://www.schemaweb.info/schema/SchemaDetails.aspx?id=62

[11] http://www.loa-cnr.it/DOLCE.html



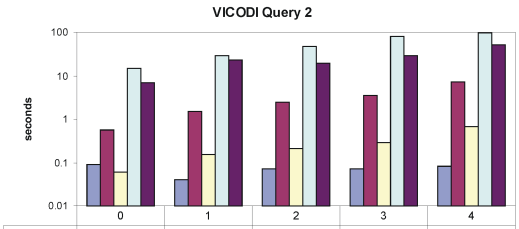




Figure 13.1: VICODI Ontology Test Results

modules. We used the DOLCE OWL version 397, up to the Common module (this includes the DOLCE-Lite, ExtDnS, Modal, and Common modules).

GALEN[12] is a medical terminology ontology developed in the GALEN project [114]. It has a very large and complex TBox, and has traditionally been used as a benchmark for terminological reasoning.

## 13.3 Querying Large ABoxes

We now show the results of our tests of querying large ABoxes.

### 13.3.1 VICODI

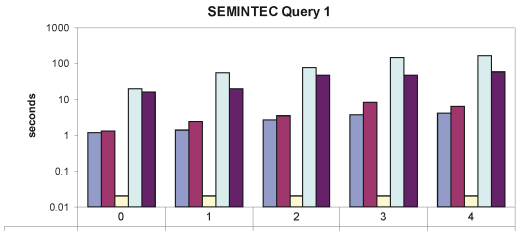
Because VICODI does not contain existential quantifiers or disjunctions, it can be converted into disjunctive datalog directly, without invoking the reduction algorithm. Hence, reasoning with VICODI requires only an efficient deductive database. From the ontology author we received the following two queries, used in the project:

$$Q_{V_1}(x) \equiv \mathit{Individual}(x)$$
$$Q_{V_2}(x,y,z) \equiv \mathit{Military\text{-}Person}(x), \mathit{hasRole}(y,x), \mathit{related}(x,z)$$

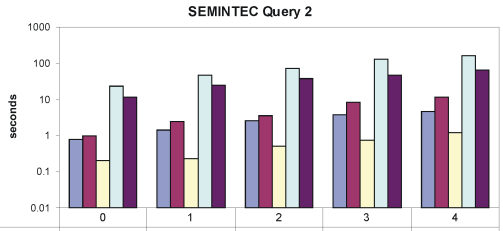
The results in Figure 13.1 show that Pellet and RACER spend the bulk of their time in checking ABox consistency by computing a completion of the ABox. Because the ontology is simple, no branch splits are performed, so the process yields a single completion representing a model. Query answering is then very fast, as it amounts to model lookup.

According to the authors of Racer, the gap in performance between Pellet and Racer should be resolved in the next release of Racer.

[12]We obtained GALEN through private communication with Ian Horrocks.




| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| KAON2 | 1.18 | 1.43 | 2.65 | 3.63 | 4.10 |
| Pellet (S) | 1.32 | 2.45 | 3.61 | 8.22 | 6.26 |
| Pellet (Q) | 0.02 | 0.02 | 0.02 | 0.02 | 0.02 |
| RACER (S) | 20.54 | 56.18 | 78.12 | 149.30 | 168.30 |
| RACER (Q) | 16.33 | 20.39 | 48.66 | 49.05 | 59.83 |




| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| KAON2 | 0.80 | 1.44 | 2.58 | 3.67 | 4.72 |
| Pellet (S) | 0.96 | 2.38 | 3.53 | 8.13 | 11.22 |
| Pellet (Q) | 0.20 | 0.23 | 0.51 | 0.72 | 1.18 |
| RACER (S) | 23.46 | 46.14 | 72.91 | 128.82 | 157.82 |
| RACER (Q) | 11.29 | 24.16 | 38.53 | 46.31 | 64.90 |

Figure 13.2: SEMINTEC Ontology Test Results

### 13.3.2 SEMINTEC

The SEMINTEC ontology is also very simple; however, it is interesting because it contains functional roles and therefore requires equality reasoning. From the ontology author we obtained the following two queries, used in the project:

$$\begin{aligned} Q_{S_1}(x) &\equiv \mathit{Person}(x) \\ Q_{S_2}(x,y,z) &\equiv \mathit{Man}(x), \mathit{isCreditCardOf}(y,x), \mathit{Gold}(y), \mathit{livesIn}(x,z), \mathit{Region}(z) \end{aligned}$$

The results of running the queries are shown in Figure 13.2. The SEMINTEC ontology is roughly of the same size as the VICODI ontology; however, the time that KAON2 takes to answer a query on SEMINTEC are one order of magnitude larger than for the VICODI ontology. This is mainly due to equality, which is difficult for deductive databases.

### 13.3.3 LUBM

LUBM is comparable in size to the VICODI and the SEMINTEC ontologies, but its TBox contains complex concepts. It uses existential quantifiers, so our reduction algorithm must be used to eliminate function symbols. Also, the ontology does not contain disjunctions or equality, so the translation yields an equality-free Horn program.

We wanted a mix of simple and complex queries, so we selected the following three queries from the LUBM Web site:

$$\begin{aligned} Q_{L_1}(x) &\equiv \mathit{Chair}(x) \\ Q_{L_2}(x,y) &\equiv \mathit{Chair}(x), \mathit{worksFor}(x,y), \mathit{Department}(y), \\ &\quad \mathit{subOrganizationOf}(y, \text{http://www.University0.edu}) \\ Q_{L_3}(x,y,z) &\equiv \mathit{Student}(x), \mathit{Faculty}(y), \mathit{Course}(z), \mathit{advisor}(x,y), \mathit{takesCourse}(x,z), \\ &\quad \mathit{teacherOf}(y,z) \end{aligned}$$

As our results from Figure 13.3 show, LUBM does not pose significant problems for KAON2; namely, the translation produces an equality-free Horn program, which

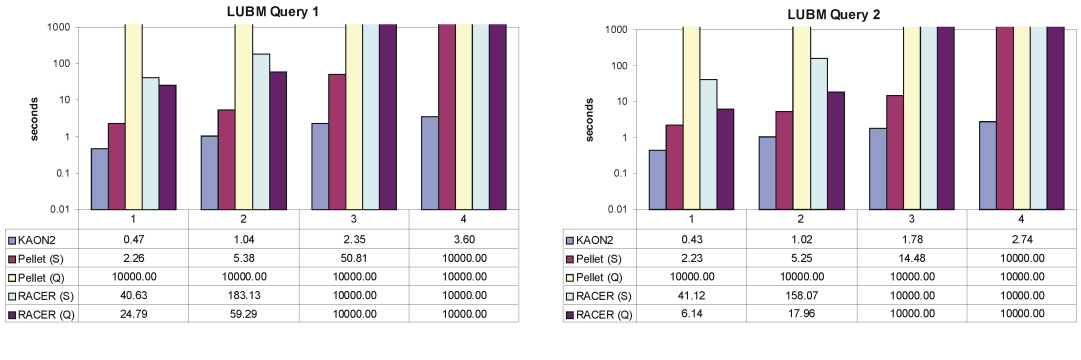
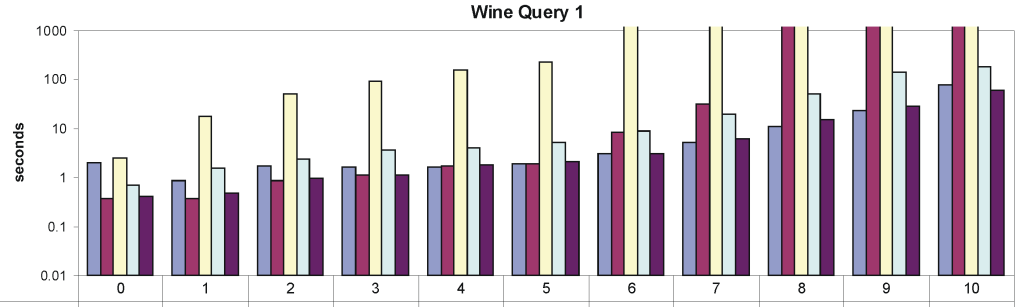



Figure 13.3: LUBM Ontology Test Results

KAON2 evaluates in polynomial time. Although LUBM is roughly of the same size as VICODI, both Pellet and Racer performed better on the latter; namely, Pellet was not able to answer any of the LUBM queries within the given resource constraints, and Racer performed significantly better on VICODI than on LUBM. We were surprised by this result: the ontology is still Horn, so an ABox completion can be computed in advance and used as a cache for query answering. By analyzing a run of Pellet on lubm_1 in a debugger, we observed that the system performs disjunctive reasoning (that is, it performs branch splits). Further investigation showed that this is due to *absorption* [66]—a well-known optimization technique used by all tableau reasoners. Namely, an axiom of the form $C \sqsubseteq D$, where $C$ is a complex concept, increases the amount of don't-know nondeterminism in a tableau because it yields a disjunction $\neg C \sqcup D$ in the label of each node. If possible, such an axiom is transformed into an equivalent *definition* axiom $A \sqsubseteq C'$ (where $A$ is an atomic concept), which can be handled in a deterministic way. The LUBM ontology contains several axioms that are equivalent to $A \sqsubseteq B \sqcap \exists R.C$ and $B \sqcap \exists R.C \sqsubseteq A$. Now the latter axiom contains a complex concept on the left-hand side of $\sqsubseteq$, so it is absorbed into an equivalent axiom $B \sqsubseteq A \sqcup \forall R.\neg C$. Whereas this is a definition axiom, it contains a disjunction on the right-hand side, and thus causes branch splits.




| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| KAON2 | 2.02 | 0.85 | 1.72 | 1.60 | 1.61 | 1.95 | 3.16 | 5.34 | 10.78 | 23.16 | 80.44 |
| Pellet (S) | 0.36 | 0.36 | 0.86 | 1.15 | 1.76 | 1.88 | 8.61 | 31.79 | 10000.00 | 10000.00 | 10000.00 |
| Pellet (Q) | 2.47 | 17.50 | 50.44 | 92.48 | 154.82 | 224.77 | 10000.00 | 10000.00 | 10000.00 | 10000.00 | 10000.00 |
| RACER (S) | 0.68 | 1.58 | 2.33 | 3.53 | 3.93 | 5.31 | 8.91 | 19.89 | 51.53 | 138.30 | 182.77 |
| RACER (Q) | 0.41 | 0.47 | 0.94 | 1.12 | 1.82 | 2.07 | 3.10 | 6.02 | 15.35 | 29.35 | 59.63 |

Figure 13.4: Wine Ontology Test Results

### 13.3.4 Wine

The Wine ontology is a fairly complex ontology, using advanced DL constructors, such as disjunctions and equality. The translation of nominals is incomplete, so we ran only the following query:

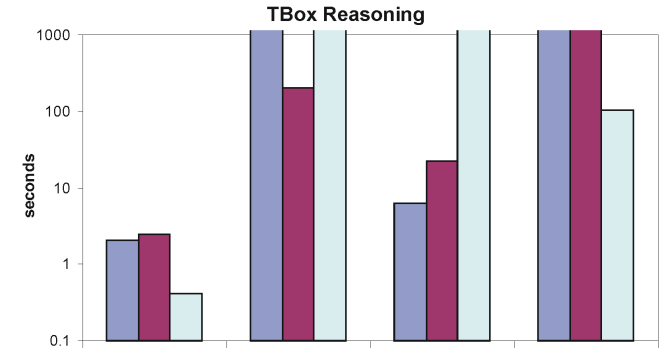
$$Q_{W_1}(x) \equiv AmericanWine(x)$$

The results from Figure 13.4 show that the ontology complexity affects the performance: wine_0 is significantly smaller than, say, lubm_1, but the time required to answer the query is roughly the same. The degradation of performance in KAON2 is mainly due to disjunctions. On the theoretical side, disjunctions increase the data complexity of our algorithm from P to NP [153]. On the practical side, the technique for answering queries in disjunctive programs used in KAON2 should be further optimized.

## 13.4 TBox Reasoning

Although TBox reasoning was not in the focus of our work, to better understand the limitations of our algorithms, we also conducted several TBox reasoning tests. In particular, we measured the time required to compute the subsumption hierarchies of Wine, DOLCE, and GALEN. Furthermore, we observed that a considerable source of complexity for KAON2 on DOLCE are the transitivity axioms, so we also performed the tests for a version of DOLCE in which all transitivity axioms were removed.

The results from Figure 13.5 show that the performance of TBox reasoning in KAON2 lags behind the performance of the state-of-the-art tableau reasoners. This should not come as a surprise: in the past decade, many techniques for optimizing




| | Wine | DOLCE | DOLCE (No Trans.) | GALEN |
|---|---|---|---|---|
| KAON2 | 2.02 | 10000.00 | 6.25 | 10000.00 |
| Pellet | 2.47 | 198.98 | 22.05 | 10000.00 |
| RACER | 0.41 | 10000.00 | 10000.00 | 102.03 |

Figure 13.5: TBox Test Results

TBox reasoning in tableau algorithms were developed; these techniques are not directly applicable to the resolution setting. Still, KAON2 can classify DOLCE without transitivity axioms, which is known to be a fairly complex ontology. Hence, we believe that developing additional optimization techniques for resolution algorithms might yield some interesting and practically useful results.

By analyzing the ontologies for which KAON2 was unable to compute the subsumption hierarchy within given resource limits, we noticed that they all contain many $\mathcal{ALCHIQ}$-closures of types 3 and 7 with the same role symbol, which generate many consequences. This explains why KAON2 is not able to classify the original DOLCE ontology, but why it works well if the transitivity axioms are removed: the transformation used to deal with transitivity introduces axioms that, when clausified, produce many closures of types 3 and 7.

# Chapter 14

# Conclusion

Motivated primarily by the prospects of reusing well-known deductive database techniques to optimize query answering in description logics, we developed several novel algorithms for reasoning with description logics related to $\mathcal{SHIQ}$.

Our algorithms can handle the description logic $\mathcal{SHIQ}(\mathbf{D})$, which is closely related to the ontology language for the Semantic Web OWL-DL. The logical underpinning of OWL-DL is actually the $\mathcal{SHOIN}(\mathbf{D})$ description logic, which differs from $\mathcal{SHIQ}(\mathbf{D})$ in that it supports nominals, but does not provide for qualified number restrictions.

The first algorithm we presented is an algorithm for checking satisfiability of $\mathcal{SHIQ}$ knowledge bases based on basic superposition, a clausal calculus for theorem proving with equality. The novel aspect is that the procedure allows clauses to contain terms of depth two, and relies on basic superposition to block certain undesirable inferences. Furthermore, it relies on subsumption to restrict the term depth, and not just the clause length. Our procedure runs in time exponential in the size of the knowledge base for unary coding of numbers, which makes it worst-case optimal. It is worth noting that the assumption on unary coding of numbers is standard in practical description logic reasoning systems. Basic superposition alone decides only a slightly weaker logic $\mathcal{SHIQ}^{-}$, so to handle $\mathcal{SHIQ}$, we extend basic superposition with the decomposition rule, for which we show soundness and completeness.

An important aspect of OWL-DL is that it provides constructs for representing concrete data, such as strings or integers. Generally, such capabilities are integrated into description logics using so-called concrete domains. Until now, reasoning with a concrete domain has been studied predominantly in the context of tableaux and automata calculi. These existing approaches to reasoning with a concrete domain are not directly applicable to clausal calculi, so we devised an algorithm for reasoning with a concrete domain in the resolution framework. This approach is general and is applicable to any clausal calculus whose completeness proof is based on the model generation method. Furthermore, we applied this approach to derive a decision procedure for $\mathcal{SHIQ}(\mathbf{D})$. Assuming unary coding of numbers, an upper bound on the arity of concrete predicates, and an exponential decision procedure for checking satisfiability

of concrete predicates, adding a concrete domain does not increase the complexity of reasoning; that is, our algorithm still runs in exponential time.

We applied the decision procedure for $\mathcal{SHIQ}(\mathbf{D})$ to obtain an algorithm for reducing a $\mathcal{SHIQ}(\mathbf{D})$ knowledge base to a disjunctive datalog program. The reduction is based on the idea of simulating the inference steps of basic superposition in disjunctive datalog. This allows reusing various optimizations for ABox query answering, such as join order optimizations or the magic sets transformation.

Based on this reduction, we showed that checking satisfiability of $\mathcal{SHIQ}(\mathbf{D})$ knowledge bases is data complete for NP. This is a somewhat surprising result, since, under the assumption that NP $\subset$ ExpTime, it is better than the combined complexity of the same problem. To further reduce data complexity, we proposed a Horn fragment of $\mathcal{SHIQ}(\mathbf{D})$, in which the capability for modeling disjunctive information is traded for polynomial data complexity. We believe that this fragment is very relevant in practice, since it is still very expressive, but provides hope for a tractable implementation.

We extended our algorithms with a simple, but useful feature. Namely, we showed that so-called DL-safe rules can freely be appended to the resulting disjunctive program. The DL-safe rules are characterized by the restriction that each variable occurring in a rule also occurs in a non-DL-atom in the rule body, which limits the applicability of rules to individuals introduced explicitly in the ABox. Since the number of such individuals is finite, adding DL-safe rules to $\mathcal{SHIQ}(\mathbf{D})$ does not cause termination problems.

We also showed that basic superposition can be used to answer conjunctive queries over $\mathcal{SHIQ}(\mathbf{D})$ knowledge bases. Using the well-known reduction of conjunctive query containment to query answering, this algorithm can be used for deciding query containment as well. Conjunctive queries provide an expressive language for querying description logic knowledge bases.

We also considered extending $\mathcal{SHIQ}(\mathbf{D})$ with metamodeling. We showed that the approach to metamodeling adopted in the Semantic Web standard OWL-Full leads to undecidability. Therefore, we proposed an alternative semantics for metamodeling, for which we give a decision procedure.

To show that our algorithms are practically relevant, we implemented them in a new DL reasoning system KAON2, and conducted a performance evaluation. For ABox query answering, our measurements show an increase of one and more orders of magnitude, when compared to the performance of existing tableaux-based systems, such as Pellet or RACER. For TBox reasoning, our system does not match the performance of tableaux-based ones; however, it is still able to solve certain nontrivial problems. Hence, we believe that developing additional optimization techniques might yield additional improvements, making resolution match and perhaps even outperform tableau-based reasoners.

For our future work, an important theoretical challenge is to extend the algorithms to handle nominals. A tableau decision procedure for $\mathcal{SHIQ}$ extended with nominals has been presented recently in [71]; it is interesting to see whether the ideas presented there can be transferred into the framework of resolution.

Another theoretical challenge is to provide a decision procedure capable of dealing with transitivity directly, without encoding transitivity axioms. Namely, practical experience has shown that dealing with constructors by an encoding usually results in suboptimal performance. Furthermore, a method for handling transitivity directly would allow using transitive roles in conjunctive queries and DL-safe rules.

Finally, an important challenge is to extend our approach by some form of nonmonotonic reasoning. Namely, recent experience has shown that features such as closed-world and default reasoning are very important in numerous applications of description logics. Furthermore, disjunctive datalog has been traditionally considered a platform for nonmonotonic reasoning. Hence, it is interesting to see whether nonmonotonic features of disjunctive datalog can be reused to provide some kind of nonmonotonic reasoning in description logics.

# References


[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison Wesley, 1995.

[2] G. Alsaç and C. Baral. Reasoning in description logics using declarative logic programming. Technical report, Arizona State University, Arizona, USA, 2002. `http://www.public.asu.edu/~cbaral/papers/descr-logic-aaai2.pdf`.

[3] H. Andréka, J. van Benthem, and I. Németi. Modal Languages and Bounded Fragments of Predicate Logic. *Journal of Philosophical Logic*, 27(3):217–274, 1998.

[4] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, January 2003.

[5] F. Baader and P. Hanschke. A Scheme for Integrating Concrete Domains into Concept Languages. In J. Mylopoulos and R. Reiter, editors, *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI '91)*, pages 452–457, Sydney, Australia, August 24–30 1991. Morgan Kaufmann Publishers.

[6] F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

[7] F. Baader and W. Snyder. Unification Theory. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 8, pages 445–532. Elsevier Science, 2001.

[8] M. Baaz, U. Egly, and A. Leitsch. Normal Form Transformations. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 5, pages 273–333. Elsevier Science, 2001.

[9] L. Bachmair and H. Ganzinger. Rewrite-based Equational Theorem Proving with Selection and Simplification. *Journal of Logic and Computation*, 4(3):217–247, 1994.

[10] L. Bachmair and H. Ganzinger. Equational Reasoning in Saturation-Based Theorem Proving. In W. Bibel and P. H. Schmidt, editors, *Automated Deduction: A*

*Basis for Applications*, volume I: Foundations, Calculi and Methods, chapter 11, pages 353–397. Kluwer, 1998.

[11] L. Bachmair and H. Ganzinger. Ordered Chaining Calculi for First-Order Theories of Transitive Relations. *Journal of the ACM*, 45(6):1007–1049, 1998.

[12] L. Bachmair and H. Ganzinger. Strict Basic Superposition. In *Proc. of the 15th Int. Conf. on Automated Deduction (CADE-15)*, volume 1421 of *LNAI*, pages 160–174, Lindau, Germany, July 5–10 1998. Springer.

[13] L. Bachmair and H. Ganzinger. Resolution Theorem Proving. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 2, pages 19–99. Elsevier Science, 2001.

[14] L. Bachmair, H. Ganzinger, C. Lynch, and W. Snyder. Basic Paramodulation. *Information and Computation*, 121(2):172–192, 1995.

[15] P. Baumgartner. An Ordered Theory Resolution Calculus. In A. Voronkov, editor, *Proc. of the 3rd Int. Conf. on Logic Programming and Automated Reasoning (LPAR '92)*, volume 624 of *LNAI*, pages 119–130, St. Petersburg, Russia, July 15–20 1992. Springer.

[16] S. Bechhofer, R. Möller, and P. Crowther. The DIG Description Logic Interface. In D. Calvanese, G. de Giacomo, and F. Franconi, editors, *Proc. of the 2003 Int. Workshop on Description Logics (DL 2003)*, volume 81 of *CEUR Workshop Proceedings*, Rome, Italy, September 5–7 2003.

[17] S. Bechhofer, R. Volz, and P. W. Lord. Cooking the Semantic Web with the OWL API. In D. Fensel, K. P. Sycara, and J. Mylopoulos, editors, *Proc. of the 2nd Int. Semantic Web Conference (ISWC 2003)*, volume 2870 of *LNCS*, pages 659–675, Sanibel Island, FL, USA, October 20–23 2003. Springer.

[18] C. Beeri and R. Ramakrishnan. On the power of magic. In *Proc. of the 6th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS '87)*, pages 269–283, San Diego, CA, USA, March 23–25 1987. ACM Press.

[19] S. Bergamaschi, S. Castano, M. Vincini, and D. Beneventano. Semantic Integration of Heterogeneous Information Sources. *Data & Knowledge Engineering*, 36(3):215–249, 2001.

[20] R. Berger. The undecidability of the dominoe problem. *Memoirs of the American Mathematical Society*, 66:1–72, 1966.

[21] A. Borgida. On the Relative Expressiveness of Description Logics and Predicate Logics. *Artificial Intelligence*, 82(1–2):353–367, 1996.

[22] A. Borgida, R. J. Brachman, D. L. McGuinness, and L. A. Resnick. CLASSIC: a structural data model for objects. *ACM SIGMOD Record*, 18(2):58–67, 1989.

[23] R. Boyer. *Locking: A Restriction of Resolution.* PhD thesis, University of Texas at Austin, TX, USA, 1971.

[24] R. J. Brachman and J. G. Schmolze. An Overview of the KL-ONE Knowledge Representation System. *Cognitive Science*, 9(2):171–216, 1985.

[25] S. Brass and U. W. Lipeck. Generalized Bottom-Up Query Evaluation. In A. Pirotte, C. Delobel, and G. Gottlob, editors, *Proc. of the 3rd Int. Conf. on Extending Database Technology (EDBT '92)*, volume 580 of *LNCS*, pages 88–103, Vienna, Austria, March 23–27 1992. Springer.

[26] C.-L. Chang and R. C.-T. Lee. *Symbolic Logic and Mechanical Theorem Proving.* Academic Press, Inc., 1997.

[27] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. DL-Lite: Tractable Description Logics for Ontologies. In M. M. Veloso and S. Kambhampati, editors, *Proc. of the 20th National Conf. on Artificial Intelligence (AAAI 2005)*, pages 602–607, Pittsburgh, PA, USA, July 9–13 2005. AAAI Press.

[28] D. Calvanese, G. De Giacomo, and M. Lenzerini. On the Decidability of Query Containment under Constraints. In *Proc. of the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS '98)*, pages 149–158, Seattle, WA, USA, June 1–3 1998. ACM Press.

[29] D. Calvanese, G. De Giacomo, and M. Lenzerini. Answering queries using views over description logics knowledge bases. In *Proc. of the 17th National Conf. on Artificial Intelligence (AAAI 2000)*, pages 386–391, Austin, TX, USA, July 30–August 3 2000. AAAI Press.

[30] D. Calvanese, G. De Giacomo, and M. Lenzerini. Conjunctive Query Containment and Answering under Description Logics Constraints. Submitted to an international journal, 2005.

[31] D. Calvanese, M. Lenzerini, and D. Nardi. Unifying Class-Based Representation Formalisms. *Journal of Artificial Intelligence Research (JAIR)*, 11:199–240, 1999.

[32] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In J. E. Hopcroft, E. P. Friedman, and M. A. Harrison, editors, *Proc. of the 9th annual ACM Symposium on Theory of Computing (STOC '77)*, pages 77–90, Boulder, CO, USA, May 2–4 1977. ACM Press.

[33] W. Chen, M. Kifer, and D. S. Warren. A Foundation for Higher-Order Logic Programming. *Journal of Logic Programming*, 15(3):187–230, 1993.

[34] C. Cumbo, W. Faber, G. Greco, and N. Leone. Enhancing the Magic-Set Method for Disjunctive Datalog Programs. In B. Demoen and V. Lifschitz, editors, *Proc. of the 20th Int. Conf. on Logic Programming (ICLP 2004)*, volume 3132 of *LNCS*, pages 371–385, Saint-Malo, France, September 6–10 2004. Springer.

[35] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and expressive power of logic programming. *ACM Computing Surveys*, 33(3):374–425, 2001.

[36] H. de Nivelle. A Resolution Decision Procedure for the Guarded Fragment. In C. Kirchner and H. Kirchner, editors, *Proc. of the 15th Int. Conf. on Automated Deduction (CADE-15)*, volume 1421 of *LNAI*, pages 191–204, Lindau, Germany, July 5–10 1998. Springer.

[37] H. de Nivelle. Splitting Through New Proposition Symbols. In R. Nieuwenhuis and A. Voronkov, editors, *Proc. of the 8th Int. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR 2001)*, volume 2250 of *LNAI*, pages 172–185, Havana, Cuba, December 3–7, 2001. Springer.

[38] N. Dershowitz and D. A. Plaisted. Rewriting. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 9, pages 535–610. Elsevier Science, 2001.

[39] F. M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. AL-log: Integrating Datalog and Description Logics. *Journal of Intelligent Information Systems*, 10(3):227–252, 1998.

[40] J. Edelmann and B. Owsnicki. Data Models in Knowledge Representation Systems: A Case Study. In C.-R. Rollinger and W. Horn, editors, *Proc. of the 10th German Workshop on Artificial Intelligence (GWAI'86) and the 2nd Austrian Symposium on Artificial Intelligence (ÖGAI'86)*, volume 124 of *Informatik-Fachberichte*, pages 69–74. Springer, Ottenstein/Niederösterreich, September 22–26 1986.

[41] T. Eiter, W. Faber, N. Leone, and G. Pfeifer. Declarative problem-solving using the DLV system. *Logic-Based Artificial Intelligence*, pages 79–103, 2000.

[42] T. Eiter, G. Gottlob, and H. Mannila. Disjunctive Datalog. *ACM Transactions on Database Systems*, 22(3):364–418, 1997.

[43] T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. A Deductive System for Non-Monotonic Reasoning. In J. Dix, U. Furbach, and A. Nerode, editors, *Proc. of the 4th Int. Conf. on Logic Programming and Non-monotonic Reasoning (LPNMR '97)*, volume 1265 of *LNAI*, pages 364–375, Dagstuhl, Germany, July 28–31 1997. Springer.

[44] T. Eiter, T. Lukasiewicz, R. Schindlauer, and H. Tompits. Combining Answer Set Programming with Description Logics for the Semantic Web. In D. Dubois,

C. A. Welty, and M.-A. Williams, editors, *Proc. of the 9th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2004)*, pages 141–151, Whistler, Canada, June 2–5, 2004 2004. AAAI Press.

[45] C. Fermüller, T. Tammet, N. Zamov, and A. Leitsch. *Resolution Methods for the Decision Problem*, volume 679 of *LNAI*. Springer, 1993.

[46] M. Fitting. *First-Order Logic and Automated Theorem Proving, 2nd Edition*. Texts in Computer Science. Springer, 1996.

[47] H. Ganzinger and H. de Nivelle. A Superposition Decision Procedure for the Guarded Fragment with Equality. In *Proc. of the 14th IEEE Symposium on Logic in Computer Science (LICS '99)*, pages 295–305, Trento, Italy, July 2–5 1999. IEEE Computer Society.

[48] H. Ganzinger, U. Hustadt, C. Meyer, and R. A. Schmidt. A Resolution-Based Decision Procedure for Extensions of K4. In M. Zakharyaschev, K. Segerberg, M. de Rijke, and H. Wansing, editors, *Proc. of the 2nd Int. Workshop on Advances in Modal Logic (AiML '98)*, pages 243–263, Uppsala, Sweden, October 16–18 2000. CSLI Publications.

[49] L. Georgieva, U. Hustadt R. A., and Schmidt. Hyperresolution for Guarded Formulae. *Journal of Symbolic Computation*, 36(1–2):163–192, 2003.

[50] B. Glimm and I. Horrocks. Handling Cyclic Conjunctive Queries. In I. Horrocks, U. Sattler, and F. Wolter, editors, *Proc. of the 2005 Int. Workshop on Description Logics (DL 2005)*, volume 147 of *CEUR Workshop Proceedings*, Edinburgh, UK, July 26-28 2005. Poster.

[51] F. Goasdoué and M.-C. Rousset. Answering Queries using Views: A KRDB Perspective for the Semantic Web. *ACM Transactions on Internet Technology*, 4(3):255–288, 2004.

[52] G. Gottlob and C. G. Fermüeller. Removing redundancy from a clause. *Artificial Intelligence*, 61(2):263–289, 1993.

[53] G. Gottlob and A. Leitsch. On the Efficiency of Subsumption Algorithms. *Journal of the ACM*, 32(2):280–295, 1985.

[54] E. Grädel, M. Otto, and E. Rosen. Two-Variable Logic with Counting is Decidable. In *Proc. of the 12th IEEE Symposium on Logic in Computer Science (LICS '97)*, pages 306–317, Warsaw, Poland, June 29–July 2 1997. IEEE Computer Society.

[55] S. Greco. Binding Propagation Techniques for the Optimization of Bound Disjunctive Queries. *IEEE Transactions on Knowledge and Data Engineering*, 15(2):368–385, 2003.

[56] C. Green. Theorem proving by resolution as a basis for question-answering systems. In B. Meltzer and D. Michie, editors, *Proc. of the 4th Annual Machine Intelligence Workshop*, pages 183–208. Edinburgh University Press, 1969.

[57] B. N. Grosof, I. Horrocks, R. Volz, and S. Decker. Description Logic Programs: Combining Logic Programs with Description Logic. In *Proc. of the 12th Int. World Wide Web Conference (WWW 2003)*, pages 48–57, Budapest, Hungary, May 20–24 2003. ACM Press.

[58] Y. Guo, Z. Pan, and J. Heflin. An Evaluation of Knowledge Base Systems for Large OWL Datasets. In S. A. McIlraith, D. Plexousakis, and F. van Harmelen, editors, *Proc. of the 3rd Int. Semantic Web Conference (ISWC 2004)*, volume 3298 of *LNCS*, pages 274–288, Hiroshima, Japan, November 7–11 2004. Springer.

[59] V. Haarslev and R. Möller. RACER System Description. In R. Goré, A. Leitsch, and T. Nipkow, editors, *Proc. of the 1st Int. Joint Conf. on Automated Reasoning (IJCAR 2001)*, volume 2083 of *LNAI*, pages 701–706, Siena, Italy, June 18–23 2001. Springer.

[60] V. Haarslev and R. Möller. Optimization Strategies for Instance Retrieval. In I. Horrocks, S. Tessaris, and J. Z. Pan, editors, *Proc. of the 2002 Int. Workshop on Description Logics (DL 2002)*, volume 53 of *CEUR Workshop Proceedings*, Toulouse, France, April 19–21 2002.

[61] V. Haarslev and R. Möller. Incremental Query Answering for Implementing Document Retrieval Services. In D. Calvanese, G. de Giacomo, and F. Franconi, editors, *Proc. of the 2003 Int. Workshop on Description Logics (DL 2003)*, volume 81 of *CEUR Workshop Proceedings*, Rome, Italy, September 5–7 2003.

[62] V. Haarslev, R. Möller, and M. Wessel. The Description Logic $ALCNH_{R+}$ Extended with Concrete Domains: A Practically Motivated Approach. In R. Goré, A. Leitsch, and T. Nipkow, editors, *Proc. of the 1st Int. Joint Conf. on Automated Reasoning (IJCAR 2001)*, volume 2083 of *LNAI*, pages 29–44, Siena, Italy, June 18–23 2001. Springer.

[63] P. Hanschke, A. Abecker, and D. Drollinger. TAXON: A Concept Language with Concrete Domains. In H. Boley and M. M. Richter, editors, *Proc. of the Int. Workshop on Processing Declarative Knowledge (PDK'91)*, volume 567 of *LNAI*, pages 411–413, Kaiserslautern, Germany, July 1–3 1991. Springer.

[64] S. Heymans and D. Vermeir. Integrating Semantic Web Reasoning and Answer Set Programming. In M. De Vos and A. Provetti, editors, *Proc. of the 2nd Int. Workshop on Answer Set Programming, Advances in Theory and Implementation (ASP'03)*, volume 78 of *CEUR Workshop Proceedings*, pages 194–208, Messina, Italy, September 26–28 2003.

[65] I. Hodkinson. Loosely Guarded Fragment of First-Order Logic has the Finite Model Property. *Studia Logica*, 70(2):205–240, 2002.

[66] I. Horrocks. *Optimising Tableaux Decision Procedures for Description Logics*. PhD thesis, University of Manchester, UK, 1997.

[67] I. Horrocks. Using an Expressive Description Logic: FaCT or Fiction? In A. G. Cohn, L. Schubert, and S. C. Shapiro, editors, *Proc. of the 6th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR '98)*, pages 636–647, Trento, Italy, June 2–5 1998. Morgan Kaufmann Publishers.

[68] I. Horrocks and P. F. Patel-Schneider. Three Theses of Representation in the Semantic Web. In *Proc. of the 12th Int. World Wide Web Conference (WWW 2003)*, pages 39–47, Budapest, Hungary, May 20–24 2003. ACM Press.

[69] I. Horrocks and P. F. Patel-Schneider. A Proposal for an OWL Rules Language. In *Proc. of the 13th Int. World Wide Web Conference (WWW 2004)*, pages 723–731, New York, NY, USA, May 17–22 2004. ACM Press.

[70] I. Horrocks and U. Sattler. Ontology Reasoning in the $\mathcal{SHOQ}(\mathbf{D})$ Description Logic. In B. Nebel, editor, *Proc. of the 7th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*, pages 199–204, Seattle, WA, USA, August 4–10 2001. Morgan Kaufmann Publishers.

[71] I. Horrocks and U. Sattler. A Tableaux Decision Procedure for $\mathcal{SHOIQ}$. In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*, pages 448–453, Edinburgh, UK, July 30–August 5 2005. Morgan Kaufmann Publishers.

[72] I. Horrocks, U. Sattler, S. Tessaris, and S. Tobies. How to decide Query Containment under Constraints using a Description Logic. In M. Parigot and A. Voronkov, editors, *Proc. of the 7th Int. Conf. on Logic for Programming and Automated Reasoning (LPAR 2000)*, volume 1955 of *LNAI*, pages 326–343, Reunion Island, France, November 11–12 2000. Springer.

[73] I. Horrocks, U. Sattler, and S. Tobies. Practical Reasoning for Very Expressive Description Logics. *Logic Journal of the IGPL*, 8(3):239–263, 2000.

[74] I. Horrocks, U. Sattler, and S. Tobies. Reasoning with Individuals for the Description Logic $\mathcal{SHIQ}$. In D. MacAllester, editor, *Proc. of the 17th Int. Conf. on Automated Deduction (CADE-17)*, volume 1831 of *LNAI*, pages 482–496, Pittsburgh, PA, USA, June 17–20 2000. Springer.

[75] I. Horrocks and S. Tessaris. Querying the Semantic Web: a Formal Approach. In I. Horrocks and J. A. Hendler, editors, *Proc. of the 1st Int. Semantic Web Conference (ISWC 2002)*, volume 2342 of *LNCS*, pages 177–191, Sardinia, Italy, June 9–12 2002. Springer.

[76] U. Hustadt. *Resolution-Based Decision Procedures for Subclasses of First-Order Logic*. PhD thesis, Universität des Saarlandes, Germany, 1999.

[77] U. Hustadt and R. A. Schmidt. Issues of Decidability for Description Logics in the Framework of Resolution. In R. Caferra and G. Salzer, editors, *Selected Papers from Automated Deduction in Classical and Non-Classical Logics*, volume 1761 of *LNAI*, pages 191–205. Springer, 1999.

[78] N. Immerman. Relational queries computable in polynomial time. *Information and Control*, 68(1–3):86–104, 1986.

[79] I.V. Ramakrishnan and R. Sekar and A. Voronkov. Term indexing. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume II, chapter 26, pages 1853–1964. Elsevier Science, 2001.

[80] W. H. Joyner Jr. Resolution Strategies as Decision Procedures. *Journal of the ACM*, 23(3):398–417, 1976.

[81] B. Kallick. A decision procedure based on the resolution method. In A. J. H. Morrell, editor, *Proc. of the Int. Federation for Information Processing Congress (IFIP '68)*, volume 1 - Mathematics, Software, pages 269–275, Edinburgh, UK, August 5–10 1968. Horth-Holland.

[82] Y. Kazakov and H. de Nivelle. A Resolution Decision Procedure for the Guarded Fragment with Transitive Guards. In D. Basin and M. Rusinowitch, editors, *Proc. of 2nd Int. Joint Conf. on Automated Reasoning (IJCAR 2004)*, volume 3097 of *LNAI*, pages 122–136, Cork, Ireland, July 4–8 2004. Springer.

[83] A. Leitsch. Deciding clause classes by semantic clash resolution. *Fundamenta Informaticae*, 18:163–182, 1993.

[84] A. Y. Levy and M.-C. Rousset. Combining Horn Rules and Description Logics in CARIN. *Artificial Intelligence*, 104(1–2):165–209, 1998.

[85] A. Y. Levy, D. Srivastava, and T. Kirk. Data Model and Query Evaluation in Global Information Systems. *Journal of Intelligent Information Systems*, 5(2):121–143, 1995.

[86] B. Löchner. Things to know when implementing LPO. In S. Schulz, G. Sutcliffe, and T. Tammet, editors, *The IJCAR 2004 Workshop on Empirically Successful First OrderReasoning (ESFOR)*, Cork, Ireland, July 4–8 2004.

[87] C. Lutz. *The Complexity of Reasoning with Concrete Domains*. PhD thesis, Teaching and Research Area for Theoretical Computer Science, RWTH Aachen, Germany, 2002.

[88] C. Lutz. Description Logics with Concrete Domains—A Survey. In P. Balbiani, N.-Y. Suzuki, F. Wolter, and M. Zakharyaschev, editors, *Proc. of the 4th Int. Workshop on Advances in Modal Logic (AiML 2002)*, pages 265–296, Toulouse, France, September 30–October 2 2003. King's College Publications.

[89] C. Lutz. NExpTime-complete Description Logics with Concrete Domains. *ACM Transactions on Computational Logic*, 5(4):669–705, 2004.

[90] C. Lutz and U. Sattler. The Complexity of Reasoning with Boolean Modal Logics. In F. Wolter, H. Wansing, M. de Rijke, and M. Zakharyaschev, editors, *Proc. of the 3rd Int. Conf. on Advances in Modal Logic (AiML 2000)*, pages 329–348, Leipzig, Germany, October 4–7 2001. World Scientific.

[91] R. M. MacGregor. Inside the LOOM Description Classifier. *SIGART Bulletin*, 2(3):88–92, 1991.

[92] W. McCune. Solution of the Robbins Problem. *Journal of Automated Reasoning*, 19(3):263–276, 1997.

[93] D. L. McGuinness and J. R. Wright. An Industrial-Strength Description-Logics-Based Configurator Platform. *IEEE Intelligent Systems*, 13(4):69–77, 1998.

[94] M. Minsky. A Framework for Representing Knowledge. In J. Haugeland, editor, *Mind Design: Philosophy, Psychology, Artificial Intelligence*, pages 95–128. MIT Press, Cambridge, MA, USA, 1981.

[95] B. Nebel. Terminological Cycles: Semantics and Computational Properties. In J. F. Sowa, editor, *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, pages 331–361. Morgan Kaufmann Publishers, San Mateo, CA, USA, 1991.

[96] R. Nieuwenhuis and A. Rubio. Theorem Proving with Ordering and Equality Constrained Clauses. *Journal of Symbolic Computation*, 19(4):312–351, 1995.

[97] R. Nieuwenhuis and A. Rubio. Paramodulation-Based Theorem Proving. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 7, pages 371–443. Elsevier Science, 2001.

[98] H. De Nivelle, R. A. Schmidt, and U. Hustadt. Resolution-Based Methods for Modal Logics. *Logic Journal of the IGPL*, 8(3):265–292, 2000.

[99] A. Nonnengart and C. Weidenbach. Computing Small Clause Normal Forms. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 6, pages 335–367. Elsevier Science, 2001.

[100] L. Pacholski, W. Szwast, and L. Tendera. Complexity Results for First-Order Two-Variable Logic with Counting. *SIAM Journal on Computing*, 29(4):1083–1117, 2000.

[101] J. Z. Pan and I. Horrocks. Extending Datatype Support in Web Ontology Reasoning. In R. Meersman and Z. Tari, editors, *Proc. of the 2002 Int. Conf. on Ontologies, Databases and Applications of SEmantics (ODBASE 2002)*, volume 2519 of *LNCS*, pages 1067–1081, Irvine, CA, USA, October 30–November 1 2002. Springer.

[102] J. Z. Pan and I. Horrocks. RDFS(FA) and RDF MT: Two Semantics for RDFS. In D. Fensel, K. P. Sycara, and J. Mylopoulos, editors, *Proc. of the 3rd Int. Semantic Web Conference (ISWC 2003)*, volume 2870 of *LNCS*, pages 30–46, Sanibel Island, FL, USA, October 20–23 2003. Springer.

[103] J. Z. Pan and I. Horrocks. Web Ontology Reasoning with Datatype Groups. In D. Fensel, K. P. Sycara, and J. Mylopoulos, editors, *Proc. of the 2nd Int. Semantic Web Conference (ISWC 2003)*, volume 2870 of *LNCS*, pages 47–63, Sanibel Island, FL, USA, October 20–23 2003. Springer.

[104] C. H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1993.

[105] B. Parsia and E. Sirin. Pellet: An OWL-DL Reasoner. Poster, In Proc. of the 3rd Int. Semantic Web Conference (ISWC 2004), Hiroshima, Japan, November 7–11, 2004.

[106] P. F. Patel-Schneider, P. Hayes, and I. Horrocks. OWL Web Ontology Language: Semantics and Abstract Syntax, W3C Recommendation, February 10 2004. `http://www.w3.org/TR/owl-semantics/`.

[107] N. Peltier. On the decidability of the PVD class with equality. *Logic Journal of the IGPL*, 9(4):601–624, 2001.

[108] D. A. Plaisted and S. Greenbaum. A Structure-Preserving Clause Form Translation. *Journal of Symbolic Logic and Computation*, 2(3):293–304, 1986.

[109] D. Poole, A. Mackworth, and R. Goebel. *Computational Intelligence: A Logical Approach*. Oxford University Press, New York, NY, USA, 1997.

[110] I. Pratt-Hartmann. Counting Quantifiers and the Stellar Fragment. Technical report, University of Manchester, UK, 2003. submitted for publishing.

[111] M. R. Quillian. Word concepts: A theory and simulation of some basic capabilities. *Behavioral Science*, 12:410–430, 1967.

[112] R. Ramakrishnan, D. Srivastava, and S. Sudarshan. Rule Ordering in Bottom-Up Fixpoint Evaluation of Logic Programs. *IEEE Transactions on Knowledge and Data Engineering*, 6(4):501–517, 1994.

[113] R. Ramesh, I. V. Ramakrishnan, and D. S. Warren. Automata-Driven Indexing of Prolog Clauses. *Journal of Logic Programming*, 23(2):151–202, 1995.

[114] A. L. Rector, W. A. Nowlan, and A. Glowinski. Goals for concept representation in the galen project. In C. Safran, editor, *Proc. of the 17th Annual Symposium on Computer Applications in Medical Care (SCAMC '93)*, pages 414–418, Washington DC, USA, November 1–3 1993. McGraw-Hill.

[115] R. Reiter. Two Results on Ordering for Resolution with Merging and Linear Format. *Journal of the ACM*, 18(4):630–646, 1971.

[116] A. Riazanov. *Implementing an Efficient Theorem Prover*. PhD thesis, University of Manchester, UK, 2003.

[117] A. Riazanov and A. Voronkov. Partially Adaptive Code Trees. In M. Ojeda-Aciego, I. P. de Guzmán, G. Brewka, and L. Moniz Pereira, editors, *Proc. European Workshop on Logics in Artificial Intelligence, European Workshop (JELIA 2000)*, volume 1919 of *LNAI*, pages 209–223, Malaga, Spain, September 29–October 2 2000. Springer.

[118] A. Riazanov and A. Voronkov. Splitting Without Backtracking. In B. Nebel, editor, *Proc. of the 7th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*, pages 611–617, Seattle, WA, USA, August 4–10 2001. Morgan Kaufmann Publishers.

[119] A. Riazanov and A. Voronkov. The design and implementation of VAMPIRE. *AI Communications*, 15(2–3):91–110, 2002.

[120] G. Robinson and L. Wos. Paramodulation and theorem-proving in first-order theories with equality. In B. Meltzer and D. Michie, editors, *Proc. of the 4th Annual Machine Intelligence Workshop*, pages 135–158. Edinburgh University Press, 1969.

[121] J. A. Robinson. A Machine-Oriented Logic Based on the Resolution Principle. *Journal of the ACM*, 12(1):23–41, 1965.

[122] J. A. Robinson. Automatic deduction with hyper-resolution. *International Journal of Computational Mathematics*, 1(3):227–234, 1965.

[123] R. Rosati. On the decidability and complexity of integrating ontologies and rules. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, 3(1):61–73, 2005.

[124] R. Rosati. Semantic and Computational Advantages of the Safe Integration of Ontologies and Rules. In F. Fages and S. Soliman, editors, *Proc. of the 3rd Int. Workshop on Principles and Practice of Semantic Web Reasoning (PPSWR 2005)*, volume 3703 of *LNCS*, pages 50–64, Dagstuhl Castle, Germany, September 11–16 2005. Springer.

[125] A. Schaerf. On the Complexity of the Instance Checking Problem in Concept Languages with Existential Quantification. *Journal of Intelligent Information Systems*, 2(3):265–278, 1993.

[126] A. Schaerf. *Query Answering in Concept-Based Knowledge Representation Systems: Algorithms, Complexity, and Semantic Issues*. PhD thesis, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", Italy, 1994.

[127] K. Schild. A Correspondence Theory for Terminological Logics: Preliminary Report. In J. Mylopoulos and R. Reiter, editors, *Proc. of 12th Int. Joint Conf. on Artificial Intelligence (IJCAI '91)*, pages 466–471, Sydney, Australia, August 24–30 1991. Morgan Kaufmann Publishers.

[128] R. A. Schmidt and U. Hustadt. A Resolution Decision Procedure for Fluted Logic. In D. McAllester, editor, *Proc. of the 17th Int. Conf. on Automated Deduction (CADE-17)*, volume 1831 of *LNAI*, pages 433–448, Pittsburgh, PA, USA, June 17–20 2000. Springer.

[129] R. A. Schmidt and U. Hustadt. A Principle for Incorporating Axioms into the First-Order Translation of Modal Formulae. In F. Baader, editor, *Proc. of the 19th Int. Conf. on Automated Deduction (CADE-19)*, volume 2741 of *LNAI*, pages 412–426, Miami Beach, FL, USA, July 28–August 2 2003. Springer.

[130] M. Schmidt-Schauß. Subsumption in KL-ONE is undecidable. In R. J. Brachman, H. J. Levesque, and R. Reiter, editors, *Proc. of the 1st Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR '89)*, pages 421–431, Toronto, Canada, May 15–18 1989. Morgan Kaufmann Publishers.

[131] M. Schmidt-Schauß and G. Smolka. Attributive Concept Descriptions with Complements. *Artificial Intelligence*, 48(1):1–26, 1991.

[132] G. Schreiber. The Web is not well-formed. *IEEE Intelligent Systems*, 17(2):79–80, 2002. Contribution to the section "Trends & Controversies: Ontologies KISSES in Standardization", edited by S. Staab.

[133] S. Schulz. E—A Brainiac Theorem Prover. *AI Communications*, 15(2–3):111–126, 2002.

[134] S. Schulz. Simple and Efficient Clause Subsumption with Feature Vector Indexing. In S. Schulz, G. Sutcliffe, and T. Tammet, editors, *The IJCAR 2004 Workshop on Empirically Successful First OrderReasoning (ESFOR)*, Cork, Ireland, July 4–8 2004.

[135] O. Shmueli. Decidability and expressiveness aspects of logic queries. In *Proc. of the 6th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS '87)*, pages 237–249, San Diego, CA, USA, March 23–25 1987. ACM Press.

[136] O. Shmueli. Equivalence of DATALOG Queries is Undecidable. *Journal of Logic Programming*, 15(3):231–241, 1993.

[137] S. S. Skiena. *The Algorithm Design Manual*. Springer, 1997.

[138] W. Snyder. On the complexity of recursive path orderings. *Information Processing Letters*, 46(5):257–262, 1993.

[139] M. E. Stickel. Automated Deduction by Theory Resolution. *Journal of Automated Reasoning*, 1(4):333–355, 1985.

[140] M. E. Stickel. The Path-Indexing Method For Indexing Terms. Technical Report 473, Artificial Intelligence Center, SRI International, Menlo Park, California, October 1989.

[141] T. Syrjänen and I. Niemelä. The Smodels System. In T. Eiter, W. Faber, and M. Truszczynski, editors, *Proc. 6th Int. Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR 2001)*, volume 2173 of *LNAI*, pages 434–438, Vienna, Austria, September 17–19 2001. Springer.

[142] T. Tammet. *Resolution Methods for Decision Problems and Finite-Model Building*. PhD thesis, Göteborg University, Sweden, 1992.

[143] S. Tessaris. *Questions and answers: reasoning and querying in Description Logic*. PhD thesis, University of Manchester, UK, 2001.

[144] S. Tobies. *Complexity Results and Practical Algorithms for Logics in Knowledge Representation*. PhD thesis, RWTH Aachen, Germany, 2001.

[145] M. Vardi. The Complexity of Relational Query Languages (Extended Abstract). In H. R. Lewis, B. B. Simons, W. A. Burkhard, and L. Landweber, editors, *Proc. of the 14th annual ACM Symposium on Theory of Computing (STOC '82)*, pages 137–146, San Francisco, CA, USA, May 5–7 1982. ACM Press.

[146] M. Y. Vardi. On the Complexity of Bounded-Variable Queries. In *Proc. of the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS '95)*, pages 266–276, San Jose, CA, USA, May 22–25 1995. ACM Press.

[147] M. Y. Vardi. Why Is Modal Logic So Robustly Decidable? In N. Immerman and P. Kolaitis, editors, *Proc. of a DIMACS Workshop on Descriptive Complexity and Finite Models*, volume 31 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 149–184, Princeton University, USA, January 14–17 1996. American Mathematical Society.

[148] R. Volz. *Web Ontology Reasoning With Logic Databases*. PhD thesis, Universität Fridericiana zu Karlsruhe (TH), Germany, 2004.

[149] C. Welty and D. Ferrucci. What's in an instance? Technical Report 94–18, RPI Computer Science, 1994.

# Relevant Publications


[150] U. Hustadt, B. Motik, and U. Sattler. Reasoning in Description Logics with a Concrete Domain in the Framework of Resolution. In R. López de Mántaras and L. Saitta, editors, *Proc. of the 16th European Conf. on Artificial Intelligence (ECAI 2004)*, pages 353–357, Valencia, Spain, August 22–27 2004. IOS Press.

[151] U. Hustadt, B. Motik, and U. Sattler. Reducing $\mathcal{SHIQ}^-$ Description Logic to Disjunctive Datalog Programs. In D. Dubois, C. A. Welty, and M.-A. Williams, editors, *Proc. of the 9th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2004)*, pages 152–162, Whistler, Canada, June 2–5, 2004 2004. AAAI Press.

[152] U. Hustadt, B. Motik, and U. Sattler. A Decomposition Rule for Decision Procedures by Resolution-based Calculi. In F. Baader and A. Voronkov, editors, *Proc. of the 11th Int. Conf. on Logic for Programming Artificial Intelligence and Reasoning (LPAR 2004)*, volume 3452 of *LNAI*, pages 21–35, Montevideo, Uruguay, March 14–18 2005. Springer.

[153] U. Hustadt, B. Motik, and U. Sattler. Data Complexity of Reasoning in Very Expressive Description Logics. In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*, pages 466–471, Edinburgh, UK, July 30–August 5 2005. Morgan Kaufmann Publishers.

[154] B. Motik. On the Properties of Metamodeling in OWL. In Y. Gil, E. Motta, V.R. Benjamins, and M. Musen, editors, *Proc. of the 4th Int. Semantic Web Conf. (ISWC 2005)*, volume 3729 of *LNCS*, pages 548–562, Galway, Ireland, November 6–10 2005. Springer.

[155] B. Motik, U. Sattler, and R. Studer. Query Answering for OWL-DL with Rules. In S. A. McIlraith, D. Plexousakis, and F. van Harmelen, editors, *Proc. of the 3rd Int. Semantic Web Conf. (ISWC 2004)*, volume 3298 of *LNCS*, pages 549–563, Hiroshima, Japan, November 7–11 2004. Springer.

[156] B. Motik, U. Sattler, and R. Studer. Query Answering for OWL-DL with rules. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, 3(1):41–60, 2005.