DIP

Data, Information and Process Integration with Semantic Web Services

FP6 – 507483

Deliverable

WP1: Ontology reasoning and querying

D1.4

Mechanisms and Algorithms for Scalable Hybrid Reasoning

Boris Motik

January 25, 2005

Summary

As described in DIP deliverables D1.1 and D1.3 [63, 61], no single logical formalism is capable of fulfilling all the requirements imposed on a practical ontology language. Rather, combining different formalisms in a hybrid framework has been identified as the most promising way to obtain a powerful ontology language suitable for modeling practical problems.

In D1.3, a framework for hybrid reasoning has been presented, providing theoretical means for integrating various formalisms into a unifying framework. The framework is based on reducing various formalisms to disjunctive datalog, which is then used as a mechanism for query answering. One component language of our framework is OWL-DL— the current standard for ontology modeling in the Semantic Web. Another important component language is Web Service Modeling Language (WSML), a language developed in WP 2 as part of Deliverable D2.7. This language is specifically designed to address the issues of Web service modeling. Hence, our framework allows interoperability between WSML and OWL-DL. On the practical side, it allows the service descriptions formalized in WSML to refer to domain vocabulary expressed in OWL-DL.

This deliverable presents the algorithms necessary to realize hybrid reasoning. The algorithms presented in this deliverable will be implemented in deliverables D1.2 and D1.7, to produce a practical tool for hybrid reasoning for the DIP community. The main idea of these algorithms is to reduce an OWL-DL ontology to a disjunctive datalog program. Furthermore, a translation of WSML ontologies into a rule-based formalism will be provided as part of D2.7. Finally, the rules obtained by translating OWL-DL and WSML ontologies can be merged and processed by any disjunctive datalog system.

Document Information

IST Project Number	FP6 – 507483	**Acronym**	DIP
Full Title	Data, Information, and Process Integration with Semantic Web Services		
Project URL	http://dip.semanticweb.org/		
Document URL			
EU Project Officer	Brian Macklin		

Deliverable	**Number**	1.4	**Title**	Mechanisms and Algorithms for Scalable Hybrid Reasoning
Work Package	**Number**	1	**Title**	Ontology reasoning and querying

Date of Delivery	**Contractual**	M12	**Actual**	20-Dec-04
Status	version 1.0		final ☐	
Nature	prototype ☐ report ☒ dissemination ☐			
Dissemination Level	public ☒ consortium ☐			

Authors (Partner)	Boris Motik (FZI)			
Resp. Author	Boris Motik		**E-mail**	motik@fzi.de
	Partner	FZI	**Phone**	+49 (721) 9654-812

Abstract (for dissemination)	This document provides details required to realize the framework for hybrid reasoning presented in D1.3. In particular, it presents the algorithm for reducing OWL-DL knowledge bases to disjunctive datalog programs. As explained in D1.3, this reduction provides the foundation for interoperability among a great number of formalisms, including, but not limiting to WSML, F-Logic, and OWL-DL.
Keywords	reasoning, query answering, (disjunctive) deductive databases, resolution, basic superposition

Version Log			
Issue Date	**Rev No.**	**Author**	**Change**
23-09-04	1	Boris Motik	First version created
13-01-05	2	Boris Motik	Edited according to comments from Jos

Project Consortium Information

Partner	Acronym	Contact
National University of Galway	NUIG	Prof. Dr. Christoph Bussler Digital Enterprise Research Institute (DERI) National University of Ireland, Galway Galway Ireland E-mail: chris.bussler@deri.ie Tel: +353 91 512460
Fundacion De La Innovacion.Bankinter	Bankinter	Monica Martinez Montes Fundacion de la Innovation. BankInter, Paseo Castellana, 29 28046 Madrid, Spain Email: mmtnez@bankinter.es Tel: 916234238
Berlecon Research GmbH	Berelcon	Dr. Thorsten Wichmann Berlecon Research GmbH, Oranienburger Str. 32, 10117 Berlin, Germany E-mail: tw@berlecon.de Tel: +49 30 2852960
British Telecommunications Plc.	BT	Dr. John Davies BT Exact (Orion Floor 5 pp12) Adastral Park Martlesham Ipswich IP5 3RE, United Kingdom Email: john.nj.davies@bt.com Tel: +44 1473 609583
Swiss Federal Institute of Technology, Lausanne	EPFL	Prof. Karl Aberer Distributed Information Systems Laboratory École Polytechnique Féderale de Lausanne Bât. PSE-A 1015 Lausanne, Switzerland E-mail : Karl.Aberer@epfl.ch Tel: +41 21 693 4679
Essex County Council	Essex	Mary Rowlatt, Essex County Council, PO Box 11, County Hall, Duke Street, Chelmsford, Essex, CM1 1LX, United Kingdom. E-mail: maryr@essexcc.gov.uk Tel: +44 (0)1245 436524
Forschungszentrum Informatik	FZI	Andreas Abecker Forschungszentrum Informatik Haid-und-Neu Strasse 10-14 76131 Karlsruhe Germany E-mail: abecker@fzi.de Tel: +49 721 96540

Institut für Informatik, Leopold-Franzens Universität Innsbruck	UIBK	Prof. Dieter Fensel Institute of computer science University of Innsbruck Technikerstr. 25 A-6020 Innsbruck, Austria Email: dieter.fensel@deri.org Tel: +43 512 5076485
ILOG SA	ILOG	Christian de Sainte Marie 9 Rue de Verdun, 94253, Gentilly, France E-mail: csma@ilog.fr Tel: +33 1 49082981
inubit AG	inubit	Torsten Schmale, inubit AG, Lützowstraße 105-106 D-10785 Berlin, Germany E-mail: ts@inubit.com Tel: +49 30726112 0
Intelligent Software Components, S.A.	iSOCO	Dr. V. Richard Benjamins, Director R&D Intelligent Software Components, S.A. Pedro de Valdivia 10 28006 Madrid, Spain E-mail: rbenjamins@isoco.com Tel. +34 913 349 797
The Open University	OU	Dr. John Domingue Knowledge Media Institute, The Open University, Walton Hall, Milton Keynes, MK7 6AA, UK E-mail: j.b.domingue@open.ac.uk Tel.: +44 1908 655014
SAP AG	SAP	Dr. Elmar Dorner SAP Research, CEC Karlsruhe SAP AG Vincenz-Priessnitz-Str. 1 76131 Karlsruhe, Germany E-mail: elmar.dorner@sap.com Tel: +49 721 6902 31
Sirma AI Ltd.	Sirma	Atanas Kiryakov, Ontotext Lab, - Sirma AI EAD, Office Express IT Centre, 3rd Floor 135 Tzarigradsko Chausse, Sofia 1784, Bulgaria E-mail: atanas.kiryakov@sirma.bg Tel.: +359 2 9768 303
Tiscali Österreich GmbH	Tiscali	Dr Dieter Haacker Tiscali Österreich Gmbh. Diefenbachgasse 35, A-1150 Vienna, Austria E-mail: Dieter.Haacker@at.tiscali.com Tel: +43 1 899 33 160

Unicorn Solution Ltd.	Unicorn	Jeff Eisenberg Unicorn Solutions Ltd, Malcha Technology Park 1 Jerusalem 96951, Israel E-mail: Jeff.Eisenberg@unicorn.com Tel.: +972 2 6491111
Vrije Universiteit Brussel	VUB	Carlo Wouters, Starlab- VUB Vrije Universiteit Brussel Pleinlaan 2, G-10 1050 Brussel, Belgium E-mail: carlo.wouters@vub.ac.be Tel.: +32 (0) 2 629 3719

Table of Contents

1 Introduction

In Deliverable D1.1 [63], several modern logical formalisms for ontology modeling have been compared. In particular, the deliverable focuses on OWL-DL— a language closely related to the family of *description logic* knowledge representation formalisms [4], and F-Logic [52] — a family of languages closely related to well-known rule-based formalisms. The main conclusion of the deliverable is that no existing logical formalism provides all features required by all use-cases in practice. Rather, complex practical problems might be solved by applying *hybrid reasoning*, where various logical formalisms are integrated in a coherent logical framework. In this way one does not need to select one particular formalism before building an application. Rather, the choice of the formalism may be delayed, and different formalisms may be chosen to solve different parts of the whole problem.

As noted in D1.1, an important goal of hybrid reasoning is to support existing formalisms as much as possible, without the need to reduce expressivity. For example, in [36] an intersection of description logic and rule-based formalisms has been presented. Unfortunately, this intersection reduces both formalisms significantly, thus removing some core features of both of them (such as existential quantifiers on the description logic side, and negation-as-failure on the rules side). This is usually true of any intersection of two formalisms.

A framework for hybrid reasoning which does not reduce component formalisms, but also does not introduce a performance penalty for using either of them, is much more desirable, as it allows using salient features of each of the component formalisms. In Deliverable D1.3 [61], such a framework for hybrid reasoning has been presented. Briefly, in order to achieve decidability, our framework reduces the exchange of consequences between component languages. The framework is based on disjunctive datalog, which serves as the underlying reasoning mechanism. Currently, the framework supports integration of (not only) the following formalisms:

- function-free Horn rules,
- F-Logic [52] (in particular, its LP variant, cf. [63]),
- Web Service Modeling Language (WSML)[1] (in particular, its Flight variant),
- the current Semantic Web ontology standard OWL-DL [71].

Most formalisms from the above list can be easily encoded in a rule-based framework. For example, most current F-Logic reasoners, such as OntoBroker[2] or FLORA-2[3] deal with F-Logic formulae by encoding them into rules. This encoding is well-known, it is roughly sketched in D1.3, and is not further elaborated in this deliverable.

Another important language developed in DIP, is WSML. It specifically addresses the problems of modeling Semantic Web Services. The language will be described in more detail in Deliverable D2.7, along with algorithms for translating WSML ontologies into rules. The combination of algorithms from this deliverable and from D2.7, yields interoperability between WSML and other formalisms. Furthermore, many

[1]http://www.wsmo.org/wsml/
[2]http://www.ontoprise.de/
[3]http://flora.sourceforge.net/

RDF(S) qualify as OWL-DL ontologies (provided that they do not use metaclasses). Hence, the algorithms in this deliverable are very relevant in practice, since they enable reusing a large number of existing domain ontologies specified in OWL-DL or RDF(S) in WSML service descriptions.

From the theoretical standpoint, integrating OWL-DL into the framework is the most difficult. Namely, OWL-DL is a decidable formalism, but provides existential quantifiers. It is easy to see that a straightforward encoding of OWL-DL into rules is undecidable. Furthermore, as explained in D1.3, a straightforward combination of OWL-DL into rules is bound to be undecidable. Rather, to deal with existential quantifiers without loosing decidability, special care must be take. This problem is addressed in D1.3 by introducing the formalism of *DL-safe* rules. Briefly, DL-safe rules do not restrict component formalisms, but restrict the interchange of consequences among them in a way that guarantees decidability.

Although DL-safe rules provide the logical basis for integration of OWL-DL with rule-based formalisms, practical algorithms for reasoning with OWL-DL combined with DL-safe rules (and thus with any other rule-based formalism) are still missing. In this deliverable, we fill this gap by providing algorithms for reasoning with DL-safe rules by reduction to the well-known formalisms of (disjunctive) deductive databases. Although in D1.3 we have shown that the combination of OWL-DL with DL-safe rules is decidable, we were unable to derive algorithms capable of handling all of OWL-DL. The main problem are *nominals* — classes containing exactly the specified set of individuals. Hence, the algorithms in this deliverable are capable of handling $\mathcal{SHIQ}(\mathbf{D})$ [44], a logical formalism obtained from OWL-DL by disallowing nominals and allowing qualified number restrictions. It is worth noting that so far, a practical algorithm for reasoning with OWL-DL is not known. Hence, all state-of-the-art description logic reasoning systems, such as FaCT[41] or RACER [37], support $\mathcal{SHIQ}(\mathbf{D})$.

The development of our algorithms is technically involved, so after introducing the necessary definitions in Chapter 2, we develop our algorithms by these steps:

- In Chapter 4 we derive an algorithm for deciding satisfiability of $\mathcal{SHIQ}$ knowledge bases by basic superposition.
- In Chapter 5 we extend the algorithm from Chapter 4 to handle *concrete* values, such as strings or integers.
- In Chapter 6 we use the algorithms from Chapter 4 and Chapter 5 to derive the translation of $\mathcal{SHIQ}(\mathbf{D})$ knowledge bases into positive disjunctive datalog programs.
- In Chapter 7 we show that DL-safe rules can be simply appended to the result of the translation obtained by algorithms from Chapter 6.

The algorithms presented in this deliverable will serve as a basis for the reasoning system developed in the context of Deliverable D1.7. This system will provide a foundation for hybrid reasoning for the DIP community. We expect this system to be used by those components of the DIP architecture which require management of ontologies and metadata, or inferencing capabilities. Some of the components that will benefit from the ontology server are:

- A repository for Web service descriptions may use the ontology server to manipulate service descriptions.

- A discovery component may use the inferencing capabilities, such as query answering or subsumption, to implement matching of service descriptions to service requests.

- A data mediation component might specify data transformation rules in a declarative manner using logic, and then use the inferencing capabilities to execute these rules.

The results from this deliverable have partially been published in [47, 46, 62].

2 Preliminaries

In this chapter we introduce terminology and recapitulate the previous work that our work is based upon. Note that this chapter is not intended to be of a tutorial nature; for details and the intuition behind the presented material, please refer to the references.

2.1 Multi-sorted First-order Logic

We assume standard definitions of first-order logic ([30] is a good text book), which we extend with multi-sorted signatures as usual. Let $\Sigma = (\mathcal{P}, \mathcal{F}, \mathcal{V}, \mathcal{S})$ be a first-order signature, where $\mathcal{P}$ is a finite set of *predicate symbols*, $\mathcal{F}$ a finite set of *function symbols*, $\mathcal{V}$ a countable set of *variables* and $\mathcal{S}$ a finite set of *sorts*. Each n-ary function symbol $f \in \mathcal{F}$ is associated with a sort signature $r_1 \times \ldots \times r_n \to r$, and each n-ary predicate symbol $P \in \mathcal{P}$ is associated with a sort signature $r_1 \times \ldots \times r_n$, where $r_{(i)} \in \mathcal{S}$. The sort of each variable is determined by the function $\mathsf{sort} : \mathcal{V} \to \mathcal{S}$.

The set of *terms* $\mathcal{T}(\Sigma)$ and the extension of the function sort to terms is defined as follows: the set $\mathcal{T}(\Sigma)$ is the smallest set such that (i) $\mathcal{V} \subseteq \mathcal{T}(\Sigma)$, and (ii) if $f \in \mathcal{F}$ has the signature $r_1 \times \ldots \times r_n \to r$ and $t_i \in \mathcal{T}(\Sigma)$ with $\mathsf{sort}(t_i) = r_i$ for $i \leq n$, 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 $P \in \mathcal{R}$ 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 $i \leq n$, 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 ϵ. If a position p_1 is a proper prefix of a position p_2, then and p_1 is *above* p_2, and p_2 is *below* p_1. A *subterm* of t at position p, denoted with $t|_p$, 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 s, denoted with $t[s]_p$, 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 Σ 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, $\varphi_1 \to \varphi_2$ 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 $(\varphi_1 \to \varphi_2) \wedge (\varphi_1 \leftarrow \varphi_2)$. A variable x in a formula φ is *free* it it does not occur under the scope of a quantifier. If φ does not have free variables, it is *closed*.

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

The *polarity* of the subformula $\varphi|_p$ at position p in a formula φ, 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 $i \in \{1, 2\}$; $\mathsf{pol}(\exists x : \varphi, 1.p) = \mathsf{pol}(\varphi, p)$; $\mathsf{pol}(\forall x : \varphi, 1.p) = \mathsf{pol}(\varphi, p)$; $\mathsf{pol}(\varphi_1 \to \varphi_2, 1.p) = -\mathsf{pol}(\varphi_1, p)$; $\mathsf{pol}(\varphi_1 \to \varphi_2, 2.p) = \mathsf{pol}(\varphi_2, p)$; $\mathsf{pol}(\varphi_1 \leftrightarrow \varphi_2, i.p) = 0$ for $i \in \{1, 2\}$.

A substitution σ is a function from $\mathcal{V}$ into $\mathcal{T}(\Sigma)$ for which $\sigma(x) \neq x$ 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 σ 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) is the substitution such that $x\sigma = x$ for each variable x and is denoted with $\{\}$. The result of *applying* a substitution σ to a term t is denoted with $t\sigma$, and 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 σ and a variable x, the substitution σ_x is defined as follows:

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

Now the application of a substitution σ to a formula φ, written $\varphi\sigma$, is defined as follows: $A(t_1, \ldots, t_n)\sigma = A(t_1\sigma, \ldots, t_n\sigma)$; $(\varphi_1 \circ \varphi_2)\sigma = \varphi_1\sigma \circ \varphi_2\sigma$ 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 τ and σ, written $\sigma\tau$, is defined as $x\sigma\tau = (x\sigma)\tau$. A substitution σ is called a *variable renaming* if it contains only mappings of the form $x \mapsto y$. A substitution σ is equivalent to θ up to variable renaming if there is a variable renaming η such that $\theta = \sigma\eta$; in such a case, θ is also equivalent to σ up to variable renaming [7]. A substitution σ is *more general* than a substitution θ if there is a substitution η such that $\theta = \sigma\eta$.

A substitution σ is a *unifier* of terms s and t if $s\sigma = t\sigma$. A unifier σ of s and t is called a *most general unifier* if, for any unifier θ of s and t, σ is either more general than θ, or it is equivalent to θ up to variable renaming. The notion of unifiers is extended to atoms in the obvious way. If a most general unifier σ 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 $\mathcal{D}$ 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 $\cdot^I$ 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 function symbol f with a signature $r_1 \times \ldots \times r_n \rightarrow r$ an interpretation function $f^I : \mathcal{D}_{r_1} \times \ldots \times \mathcal{D}_{r_n} \rightarrow \mathcal{D}_r$. 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 as B_x, is a variable assignment which assigns 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: 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 φ under I and B, written $\varphi^{I,B}$, is defined as: $[\top]^{I,B} = \mathit{true}$, $[\bot]^{I,B} = \mathit{false}$, $[A(t_1, \ldots, t_n)]^{I,B} = \mathit{true}$ if and only if $(t_1^{I,B}, \ldots, t_n^{I,B}) \in A^I$; $[\neg\varphi]^{I,B} = \neg\varphi^{I,B}$; $[\varphi_1 \circ \varphi_2]^{I,B} = \varphi_1^{I,B} \circ \varphi_2^{I,B}$ for $\circ \in \{\wedge, \vee\}$; $[\exists x : \varphi]^{I,B} = \mathit{true}$ 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 φ is closed, then $\varphi^{I,B}$ does not depend on B, so we write φ^I instead. For a closed formula φ, an interpretation I is a *model* of φ, written $I \models \varphi$, if $\varphi^I = \mathit{true}$. A closed formula φ is *valid*, written $\models \varphi$, if $I \models \varphi$ for all interpretations I; φ is *satisfiable* if $I \models \varphi$ for at least one interpretation I; and φ is *unsatisfiable* if an interpretation I, such that $I \models \varphi$, does not exist. A closed formula φ_1 *entails* a formula φ_2, written $\varphi_1 \models \varphi_2$, if $I \models \varphi_2$ for each interpretation I for which $I \models \varphi_1$. It is well-known that $\varphi_1 \models \varphi_2$ if and only if $\varphi_1 \wedge \neg\varphi_2$ is unsatisfiable. Formulae φ_1 and φ_2 are *equisatisfiable* if φ_1 is satisfiable if and only if φ_2 is satisfiable.

Let φ be a closed first-order formula, and Λ a set of positions in φ. Then $\mathsf{Def}_\Lambda(\varphi)$ is the *definitional normal form* of φ with respect to Λ 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 φ, the variables $x_1, \ldots, x_n$ are the free variables of $\varphi|_p$, and $\circ$ is $\rightarrow$ 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 [73, 8, 68] that, for any Λ, φ and $\mathsf{Def}_{\Lambda}(\varphi)$ are equisatisfiable, and that $\mathsf{Def}_{\Lambda}(\varphi)$ can be computed in polynomial time.

Let φ be a formula and p a position in φ, 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 ψ. Then $\varphi[\psi\{x \mapsto f(x_1,\ldots,x_n)\}]_p$, where f is a new function symbol not occurring in φ, is a formula obtained by *skolemization* of φ at position p. With $\mathsf{sk}(\varphi)$ we denote the formula obtained from φ by skolemization at all positions where skolemization is possible. Usually, we assume that $\mathsf{sk}(\varphi)$ is computed by *outer skolemization*, where a position p is skolemized before any position below p. It is well-known that φ and $\mathsf{sk}(\varphi)$ are equisatisfiable [30].

The subset of ground terms of $\mathcal{F}(\Sigma)$ is called the *Herbrand universe* HU of Σ. 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) $\mathcal{D}_r = HU_r$ and (ii) function symbols are interpreted by themselves, i.e. for each $f \in \mathcal{F}$ and $t_i \in HU$, we have $f^I(t_1,\ldots,t_n) = f(t_1,\ldots,t_n)$. The Herbrand base HB of Σ is the set of all ground atoms built over the Herbrand universe of Σ. A Herbrand interpretation I can equivalently be considered a subset of the Herbrand base, i.e. $I \subseteq HB$. A formula φ is satisfiable if and only if $\mathsf{sk}(\varphi)$ is satisfiable in a Herbrand interpretation [30] (this holds for the multi-sorted case as well).

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 non-negative integers. A multiset is finite if $M(x) \neq 0$ for a finite number of x; in the rest, we consider only finite multisets. M is empty, written $M = \emptyset$, if $M(x) = 0$ for all $x \in N$. 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 $M_2 \subseteq M_1$. Furthermore, the union of M_1 and M_2 is defined as $(M_1 \cup M_2)(x) = M_1(x) + M_2(x)$, the intersection is defined as $(M_1 \cap M_2)(x) = \min(M_1(x), M_2(x))$, and the difference is defined 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 $\neg A$. 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. A clause C is semantically equivalent to $\forall \mathbf{x} : C$, where $\mathbf{x}$ is the vector of 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 non-ground 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 φ, $\mathsf{Cls}(\varphi)$ is the set of clauses obtained by *clausification* of φ, i.e. by transforming $\mathsf{sk}(\varphi)$ into conjunctive normal form by exhaustive application of well-known logical equivalences. It is well-known that φ is satisfiable if and only if $\mathsf{Cls}(\varphi)$ is satisfiable in a Herbrand model. A clause C is *safe* if each variable x occurring in any literal of C occurs also in a negative literal of C.

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 and c, 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 $D \times D$. An *inverse* relation of 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) *symmetric* if $R^{-} \subseteq R$; (iii) *asymmetric* if $(x, y) \in R$ implies $(y, x) \notin R$; (iv) *antisymmetric* if $(x, y) \in R$ and $(y, x) \in R$ implies $x = y$; (v) *transitive* if $(x, y) \in R$ and $(y, z) \in R$ implies $(x, z) \in R$; and (vi) *total* if, for any two $x, y \in D$, either $R(x, y)$ or $R(y, x)$ or $x = y$. A $\circ$-closure, written R°, where $\circ$ can be a combination of reflexive, symmetric or transitive, is the smallest relation on D fulfilling the property $\circ$ such that $R \subseteq R^{\circ}$. 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 tuples in R. An object $\alpha \in D$ is in *normal form* w.r.t. R if R does not contain an element (α, β), for any β. An object β is a normal form of α w.r.t. R if β is in normal form w.r.t. R and $(\alpha, \beta) \in R^{*}$. For a general relation R, an object can have any number of normal forms.

A *partial ordering* $\succeq$ over D is a reflexive, antisymmetric and transitive relation on D. A *strict ordering* $\succ$ over D is an irreflexive and transitive relation on D. A *multiset extension* of a strict ordering $\succ$ to multisets on D, written $\succ_{mul}$, is defined as follows: $M \succ_{mul} N$ if $M \neq N$ and if $N(x) > M(x)$ for some x, then there is some $y \succ x$ for which $M(y) > N(y)$. It is well-known that for finite multisets, if $\succ$ is total, then $\succ_{mul}$ is total as well.

A *term ordering* is an ordering where $D = \mathcal{T}(\Sigma)$, for some multi-sorted first-order signature Σ. A term ordering $\succ$ is *stable under substitutions* if for all terms s and t and all substitutions sigma σ, $s \succ t$ implies $s\sigma \succ t\sigma$; it is *stable under contexts* if for all terms s, t and u and all positions p, $s \succ t$ implies $u[s]_p \succ u[t]_p$; 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) [23, 6] is a term ordering induced over a precedence on function symbols $>_P$. Each LPO has the subterm property and, if $>_P$ is total, then LPO 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) $f >_P g$ 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

A good text-book introduction to rewrite systems is [6]; an overview of the theory of rewrite systems can be found in [23]. 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, written as $\Rightarrow_R$, 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 σ 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 $t \Rightarrow_R^* u$. If $\Downarrow_R$ and symmetric-reflexive-transitive closure of $\Rightarrow_R$ coincide, then R is *confluent*. For a confluent well-founded rewrite system R, each element α has a unique normal form w.r.t. $\Rightarrow_R$, which is denoted as $\mathsf{nf}_R(\alpha)$.

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 Basic Superposition Calculus

Paramodulation is the fundamental techniques for theorem proving with equality. In order to improve its performance, in [9] a refinement of paramodulation known as *superposition* has been presented, where ordering restrictions restrict certain unnecessary inferences. However, further optimizations of paramodulation and superposition were presented in [14], by adding a so called “basicness” restriction. These optimizations are very general, but a simplified version, called basic superposition, may be found in [10, 12]. A very similar calculus, based on an inference model with constrained clauses, was presented in [65].

The idea of the basic superposition calculus 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 [60]. 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 original clause set with variable arguments. Hence, for any term $f(t)$, if t is not a variable, it was introduced by a previous unification step.

It is a common practice in equational theorem proving to consider logical theories consisting of the equality predicate exclusively. This simplifies the theoretical treatment without losing generality. Literals $P(t_1, \ldots, t_n)$, where P is not the equality predicate, are encoded as $P(t_1, \ldots, t_n) \approx \top$, so predicate symbols actually become 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: all predicate symbols and the symbol $\top$ are of a sort different from the sort of function symbols and variables. In the rest, $P(t_1, \ldots, t_n)$ is considered a syntactic shortcut for $P(t_1, \ldots, t_n) \approx \top$. To avoid ambiguity, we use the following terminology: first-order terms (function symbols) obtained by the encoding are called $\mathcal{E}$-terms ($\mathcal{E}$-function symbols); the word “predicate symbol” refers to $\approx$ and the $\mathcal{E}$-function symbols corresponding to predicate symbols before encoding, while the word “function symbol” refers to $\mathcal{E}$-function symbols corresponding to function symbols before encoding. For example, $P(x, f(x))$ is an $\mathcal{E}$-term containing $\mathcal{E}$-function symbols P and f; however, P is a predicate symbol, f is a function symbol, and only x and $f(x)$ 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 inference rules are formulated by breaking a clause into two parts: (i) the *skeleton* clause C and (ii) the substitution σ representing the cumulative effects of previous unifications. These two components together are called a *closure*, written as $C \cdot \sigma$, which 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*[1] by $[\,]$. Any position at or below a marked position is called a *substitution position*. Basic superposition can be summarized as a calculus where superposition into a substitution position is not necessary.

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.

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

Note that all variable positions are always marked, so we usually do not show this for readability purposes. 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 the following conditions are satisfied:

- the substitution σ does not contain trivial mappings of the form $x \mapsto y$ and
- 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 σ violating some of the conditions above, then let σ' be $\sigma \setminus \{x \mapsto t\}$, and replace the closure 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 example above:

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

Basic Superposition Parameters. The basic superposition calculus is parameterized with a term ordering $\succ$ and a selection function. An *admissible* term ordering $\succ$ is a reduction ordering total on ground terms such that $\top$ is the smallest element. An ordering $\succ$ can be extended to an ordering on literals (ambiguously denoted also with $\succ$) by identifying each positive literal $s \approx t$ with a multiset $\{\{s\}, \{t\}\}$, each negative literal $s \not\approx t$ with a multiset $\{\{s, t\}\}$, and comparing these multisets by 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 $L \in C$, $L \cdot \sigma$ is (strictly) maximal in $C \cdot \sigma$ if and only if it is (strictly) maximal with respect to $(C \setminus L) \cdot \sigma$.

A *selection function* selects a (possibly empty) subset of negative literals in a closure. There are no other constraints on the selection function.

[1]In [14], framing was used for marked positions. We decided to use a different notation, because framing introduced problems with the text layout. Our notation should not be confused with the notation for modalities in multi-modal logic.

Inference Rules. The basic superposition is a refutation calculus: for N a set of closures saturated up to redundancy, N is unsatisfiable if and only if it contains the empty closure. Intuitively, "saturated up to redundancy" means that any further inference with closures from N produces a closure that has already been derived. To present the rules of basic superposition, we make a technical assumption that all premises are variable-disjoint, and that all premises 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) σ 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) $\neg B_i \cdot \theta$ are selected, or nothing is selected, $i = 1$ and $\neg B_1 \cdot \theta$ is maximal w.r.t. $D \cdot \theta$.

In the ordered hyperresolution inference rule, the closures E_i are called the *electrons* or the *side premises*, and the closure N is called the *nucleus* or the *main premise*. In [14, 65] $\mathcal{BS}$ has been presented without the hyperresolution rule. However, as noted in [9], hyperresolution is actually a "macro": it combines the effects of n negative superpositions of $(A_i \approx \top) \cdot \rho$ from E_i into $(B_i \not\approx \top) \cdot \rho$ of N, resulting in $(\top \not\approx \top) \cdot \theta$, which is immediately eliminated by reflexivity resolution. Furthermore, notice that a positive superposition of a main premise into a positive literal $(B \approx \top) \cdot \rho$ results in a tautology $(\top \approx \top) \cdot \theta$, which can be eliminated. Hence, ordered hyperresolution captures all inferences from several premises involving literals with predicates other than $\approx$. One might also define ordered factoring, which combines equality resolution on $(C \vee A \approx \top \vee B \approx \top) \cdot \rho$ with reflexivity resolution. We decided not to do this to keep the presentation simpler.

Completeness of Basic Superposition. We now briefly overview the completeness proof of the basic superposition. We base our presentation on the proof by Nieuwenhuis and Rubio from [65, 66], 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 this ordering on closures also by $\succ$. Observe that, since the literal ordering is total on ground literals, the closure ordering is total on ground closures.

Let $C \cdot \sigma$ be a closure and τ a ground substitution. The set of *succeedent-top-left variables* of $C \cdot \sigma$ w.r.t. τ, written as $\mathsf{stlvars}(C \cdot \sigma, \tau)$, is defined as 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 τ a ground substitution. A variable x occurring in the skeleton C of $C \cdot \sigma$ is *variable irreducible w.r.t.* R if (i) $x\sigma\tau$ is irreducible by rewrite rules in R or (ii) for all $x \approx s \in C$, we have $x \in \mathsf{stlvars}(C \cdot \sigma, \tau)$ and $x\sigma\tau$ 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 $\mathsf{irred}_R(N)^{\prec D}$ ($\mathsf{irred}_R(N)^{\preceq D}$) be the subset of closures of $\mathsf{irred}_R(N)$ smaller than (or equal to) a ground closure D (w.r.t. the ordering $\prec$ on closures).

Let ξ by a $\mathcal{BS}$ inference with premises $D_1 \cdot \sigma$ and $D_2 \cdot \sigma$ and a conclusion $C \cdot \rho$. For a rewrite system R and a ground substitution τ such that $\xi\tau$ is a ground instance of ξ, $\xi\tau$ 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 τ such that $C \cdot \sigma\tau$ is variable irreducible w.r.t. R, we have $R \cup \mathsf{irred}_R(N)^{\preceq C \cdot \sigma\tau} \models C \cdot \sigma\tau$. An inference ξ 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 τ such that $\xi\tau$ is a variable irreducible ground instance of ξ w.r.t. R, $R \cup \mathsf{irred}_R(N)^{\prec D} \models C \cdot \rho\tau$, where $D = \max(D_1 \cdot \sigma\tau, D_2 \cdot \sigma\tau)$. 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 $C \cdot \rho \in N$, ρ 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, 65], one may 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 also well-constrained. Since $R_N \subseteq R_N{}^*$, it follows that $R_N{}^* \models N$.

Redundancy Elimination. Based on the general redundancy notion for basic superposition, several effective *redundancy elimination rules* have been presented in [14], providing means for deleting certain closures or replacing them with simpler ones, without jeopardizing completeness. We overview the most important rules next.

A closure $C \cdot \sigma$ is *reduced modulo substitution* η *relative to* a closure $D \cdot \theta$ if, for each rewrite systems R and each ground substitution τ, $C \cdot \sigma\eta\tau$ 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 of them involves the notion of η-*domination*: for two terms $s \cdot \sigma$ and $t \cdot \theta$, we say that s is η-dominated by t, written $s \cdot \sigma \sqsubseteq_\eta t \cdot \theta$, if and only if $s\sigma\eta = t\theta$ and, whenever some variable x from σ occurs in s at position p, then p is in t at or below 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 $s\sigma\eta = t\theta$. Furthermore, each marked position from s can be overlaid at or inside a marked position of t, so $s \cdot \sigma \sqsubseteq_\eta t \cdot \theta$.

This notion can be extended to literals: $(s \approx t) \cdot \sigma \sqsubseteq_\eta (w \approx v) \cdot \theta$ 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 η-dominate a negative literal and vice versa. The extension to closures is performed like this: $C \cdot \sigma \sqsubseteq_\eta D \cdot \theta$ 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 η. It may happen that, for some η, it holds that $L'\sigma\eta = L\theta$, but not $L' \cdot \sigma \sqsubseteq_\eta L \cdot \theta$. One can make $L' \cdot \sigma$ reduced relative to $L \cdot \theta$ by replacing $L' \cdot \sigma$ with a retraction $L'' \cdot \sigma'$ by instantiating those positions from L that do not overlay into a substitution position of L'. In this way, the application of a simplification or deletion rule may be enabled, while retracting from σ as little information as possible.

With these notions we can finally present the simplification rules. Closure $C \cdot \sigma$ is a *basic subsumer* of $D \cdot \theta$ if there is a substitution η such that $C\sigma\eta \subseteq D\theta$ and $C \cdot \sigma$ is reduced relative to $D \cdot \theta$ modulo η. Additionally, if $C \cdot \sigma$ has fewer literals than $D \cdot \theta$, then $D \cdot \theta$ may 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 is called *duplicate literal deletion*.

A closure $C \cdot \sigma$ can be deleted if $C\sigma$ is a tautology, meaning that $\models C\sigma$. This is called *tautology deletion*. Testing whether $C\sigma$ is a tautology requires itself theorem proving. However, the following simple syntactic checks are effective in practice: $C \cdot \sigma$ is a 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 $(s \approx t) \cdot \sigma$ with $s\sigma = t\sigma$.

A closure $(C \vee x \not\approx s) \cdot \sigma$, where $x\sigma \succ s\sigma$ 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 time polynomial in the number of literals. It is well-known that the subsumption check is NP-complete in the number of literals [33]. Moreover, η-domination can be checked in polynomial time. Furthermore, the complexity of basic tautology deletion is determined by the complexity of checking ordering constraints. Finally, two terms s and t can be compared by the lexicographic path ordering in time $\mathcal{O}(|s| \cdot |t|)$ [56, 83].

Example. We now give several examples on how to apply the inference rules of basic superposition. We first consider a resolution inference as specified below, where we assume that $\mathcal{BS}$ is parameterized by an ordering where $R(x, f(x))$ is maximal in the first, and where $\neg R(x, y)$ is selected in the second premise. The $\mathcal{E}$-terms on which the inference rule takes place are denoted like this. The closure on the left is the side premise, whereas the closure on the right 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 then apply it to the premises. By doing so, the literals on which the resolution takes place become semantically identical to $R(x, f(x))$, so resolution can be performed. However, notice that we actually apply σ to the substitution part of the premises, so the substitution part effectively accumulates the terms introduced by unification. After applying the 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.

$$\begin{array}{cc}
C(x) \vee R(x, f(x)) \cdot \{\} & \neg D(x) \vee \neg R(x, y) \vee E(y) \cdot \{\} \\
\Downarrow & \Downarrow \\
C(x) \vee R(x, f(x)) \cdot \{\} & \neg D(x') \vee \neg R(x', y) \vee E(y) \cdot \{\} \\
\Downarrow & \Downarrow \\
C(x) \vee R(x, f(x)) \cdot \{\} & \neg D(x') \vee \neg R(x', y) \vee E(y) \cdot \{x' \mapsto x, y \mapsto f(x)\} \\
\hline
\multicolumn{2}{c}{C(x) \vee \neg D(x') \vee E(y) \cdot \{x' \mapsto x, y \mapsto f(x)\}} \\
\multicolumn{2}{c}{\Downarrow} \\
\multicolumn{2}{c}{C(x) \vee \neg D(x) \vee E(y) \cdot \{y \mapsto f(x)\}}
\end{array}$$

The inference above is written by explicitly showing the skeleton and the substitution part of a closure. Below we give the same inference but written in the convenient notation, where terms occurring at positions of skeleton variables are marked. Notice that in second step the term $f(x)$ in the literal $E([f(x)])$ is introduced by unification, and is therefore marked.

$$\begin{array}{cc}
C(x) \vee R(x, f(x)) & \neg D(x) \vee \neg R(x, y) \vee E(y) \\
\Downarrow & \Downarrow \\
C(x) \vee R(x, f(x)) & \neg D(x') \vee \neg R(x', y) \vee E(y) \\
\Downarrow & \Downarrow \\
C(x) \vee R(x, f(x)) & \neg D(x) \vee \neg R(x, [f(x)]) \vee E([f(x)]) \\
\hline
\multicolumn{2}{c}{C(x) \vee \neg D(x) \vee E([f(x)])}
\end{array}$$

Next we give an example of a positive superposition inference. We assume that no literal in either of the premise is selected; 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 $f(x) \succ g(x)$. Superposition is performed from y_1 into $f(y)$, which is shown by marking the $\mathcal{E}$-terms at which the inference is performed like this. To perform the inference, 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 finally perform the superposition inference, after which we remove from the substitution all mappings for variables which do not occur in the skeleton. Observe that the literal $C(f(y)) \cdot \{y \mapsto f(x)\}$ is semantically 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.

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

Below is the same inference written using the convenient notation. The above requirement that superposition into positions corresponding to skeleton variables actually means that superposition into or below marked terms is not allowed. Hence, since the inner $f(x)$ in $E(f([f(x)]))$ is marked, superposition into it is not allowed.

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

We finish with an example of closure subsumption. Let $C_1 = C(x) \vee D(f(x))$ 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 $\eta = \{x \mapsto g(y)\}$, since (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 a literal in $C_1\eta$ can be "overlaid" into a marked subterm in a literal in C_2. On the contrary, let $C_3 = C([g(y)]) \vee D(f(g(y))) \vee E(h(y))$. In this case, C_1 does not subsume C_3 by η: 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. It is easy to see that C_1 does not subsume C_3 under any substitution.

2.5 Splitting

In some proofs we apply an additional splitting inference rule, presented below. It is borrowed from the semantic tableau calculus and performs an explicit case analysis. If some closure consists of two parts not having variables in common, one may separately assume that either part is true. If unsatisfiability is proved in both cases, the initial closure set is evidently unsatisfiable. Each of the cases introduced by the splitting rule is called a *branch*.

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.

2.6 Disjunctive Datalog

We now briefly present the syntax and semantics of disjunctive datalog. This presentation is standard and may be found in [26, 35].

Let Σ be a first-order signature such that (i) for each function symbol $f \in \mathcal{F}(\Sigma)$ the arity of f is zero 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 Σ. 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 $\mathsf{head}(r) = \{A_i\}$ is called the *rule head*, whereas the set of atoms $\mathsf{body}(r) = \{B_i\}$ is called the *rule body*. A ground rule with an empty body is called a *fact*.

Typical definitions of a disjunctive datalog program, e.g. from [26, 35], allow negated atoms in the body. This negation is usually non-monotonic, and is different from negation in first-order logic. Our approach produces only positive disjunctive datalog programs, so we omit non-monotonic negation from the definitions.

The semantics of disjunctive datalog programs is defined as follows. 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 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. A ground atom A is true in an interpretation M if $A \in M$; A is false in M if $A \notin M$. An interpretation M is a *model* of P if, for each rule $r \in \mathsf{ground}(P, HU)$, if $\mathsf{body}(r) \subseteq M$, then $\mathsf{head}(r) \cap M \neq \emptyset$ and if all atoms from M with the $\approx$ predicate yield a congruence relation (i.e. a relation that is reflexive, symmetric, transitive, and, for any predicate symbol $R \in \mathcal{P}(\Sigma)$, $R(\ldots, a, \ldots) \in M$ and $a \approx b \in M$ implies $R(\ldots, b, \ldots) \in M$).

A model M is *minimal* if no subset of M is a model. The semantics of P is the set of all minimal models of P, which is 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 all minimal models of the program contain A; A is a *brave answer* of P, written $P \models_b A$, if at least one minimal model of the program contains A. First-order entailment coincides with cautious entailment for positive ground atoms.

The size of a rule r is defined as $|r| = 1 + \sum_{1 \le i \le n} |A_i| + \sum_{1 \le j \le m} |B_j|$, 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.

3 Introduction to Description Logics

In this chapter we give a formal definition of the syntax and the semantics of description logics, and of interesting inference problems. In Section 3.1 we introduce the basic description logic $\mathcal{SHIQ}$, which we extend in Section 3.2 to $\mathcal{SHIQ}(\mathbf{D})$ by adding so-called *datatypes* to enable representing concrete values, such as strings or integers.

3.1 Description Logic $\mathcal{SHIQ}$

The syntax and the semantics of the description logic $\mathcal{SHIQ}$ are defined as follows.

Definition 3.1.1. *Let N_{R_a} be the set of abstract role names. The set of $\mathcal{SHIQ}$* abstract roles *is the set $N_{R_a} \cup \{R^- \mid R \in N_{R_a}\}$. 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 $\sqsubseteq^*$ denote the reflexive-transitive closure of $\sqsubseteq$. A role R is transitive *if $\mathsf{Trans}(R) \in KB_\mathcal{R}$; R is* simple *if there is no role S such that $S \sqsubseteq^* R$ and S is transitive; R is* complex *if it is not simple.*

Let N_C be a set of atomic *concept names. 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 name $A \in N_C$ 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 $\neg C$, $C \sqcap D$, $C \sqcup D$, $\exists R.C$, $\forall R.C$, $\leq n\, S.C$, $\geq n\, S.C$, are also $\mathcal{SHIQ}$ concepts. Concepts from N_C are called* atomic *concepts; all other ones are called* complex*. Possibly negated atomic concepts are called* literal *concepts.*

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 individual names*. 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, since OWL-DL lacks the unique name assumption (UNA), 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, cf. [4, page 60]. Second, usual definitions do not provide for ABox axioms involving negative roles. We allow such assertions as they allow checking entailment of ground role facts.

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

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 α denotes 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 $KB \cup \{C(\alpha)\}$ is satisfiable.*
- Subsumption. *A concept C is subsumed by a concept D with respect to KB if and only if $\pi(KB) \models \pi(C \sqsubseteq D)$. This is the case if and only if $KB \cup \{(C \sqcap \neg D)(\alpha)\}$ is unsatisfiable.*
- Instance checking. *An individual a is an instance of a concept C with respect to KB 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 individuals a and b with respect to KB 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 domain set $\triangle^I$ and an interpretation function $\cdot^I$, which assigns to each individual a an element a^I, to each atomic concept A a set $A^I \subseteq \triangle^I$, and to each role R a relation $R^I \subseteq \triangle^I \times \triangle^I$. The semantics of complex concepts and axioms is given in Table 3.2, where C and D are concepts, R and S are roles and a and b are individuals. These two semantics are equivalent, as first shown by Borgida in [17]. We now define several restrictions of $\mathcal{SHIQ}$, which we consider in our work.

Definition 3.1.3. *For a knowledge base KB, a role R is called* very simple *if no role S exists, such that $S \sqsubseteq R \in KB_{\mathcal{R}}$. Description logic $\mathcal{SHIQ}^-$ has the same syntax and semantics as $\mathcal{SHIQ}$, with the additional syntactical restriction that 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 $\mathcal{SHIQ}$ ($\mathcal{SHIQ}^-$), which does not allow transitivity axioms to occur in an RBox. $\mathcal{ALC}$ is the fragment of $\mathcal{ALCHIQ}$ which does not provide for role hierarchies, inverse roles and qualified number restrictions.

Often, means for comparing various description logics is needed. The following definition can be used for these purposes:

Definition 3.1.4. *Let $\mathcal{L}$, $\mathcal{L}_1$ and $\mathcal{L}_2$ be different description logics. Then* (i) *$\mathcal{L}$ is a* superset *of $\mathcal{L}_1$ if $\mathcal{L}$ provides all constructs of $\mathcal{L}_1$;* (ii) *$\mathcal{L}$ is a* subset *of $\mathcal{L}_2$ if all constructs of $\mathcal{L}$ are provided in $\mathcal{L}_2$; and* (iii) *$\mathcal{L}$ is* between *$\mathcal{L}_1$ and $\mathcal{L}_2$ if it is a superset of $\mathcal{L}_1$ and a subset of $\mathcal{L}_2$.*

Notice that, by Definition 3.1.1, the relation $\sqsubseteq^*$ can cyclic in general. In [86] it has been shown that a $\mathcal{SHIQ}$ knowledge base KB with a cyclic role hierarchy can be reduced to a knowledge base KB' with an acyclic role hierarchy, as follows. 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 Γ, we

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

$$
\begin{array}{rcl}
\hline
\multicolumn{3}{c}{\text{Mapping Concepts to FOL}} \\
\hline
\pi_y(\top, X) & = & \top \\
\pi_y(\bot, X) & = & \bot \\
\pi_y(A, X) & = & A(X) \\
\pi_y(\neg C, X) & = & \neg \pi_y(C, X) \\
\pi_y(C \sqcap D, X) & = & \pi_y(C, X) \wedge \pi_y(D, X) \\
\pi_y(C \sqcup D, X) & = & \pi_y(C, X) \vee \pi_y(D, X) \\
\pi_y(\forall R.C, X) & = & \forall y : R(X, y) \rightarrow \pi_x(C, y) \\
\pi_y(\exists R.C, X) & = & \exists y : R(X, y) \wedge \pi_x(C, y) \\
\pi_y(\leq n\, R.C, X) & = & \forall y_1, \ldots, y_{n+1} : \bigwedge R(X, y_i) \wedge \bigwedge \pi_x(C, y_i) \rightarrow \bigvee y_i \approx y_j \\
\pi_y(\geq n\, R.C, X) & = & \exists y_1, \ldots, y_n : \bigwedge R(X, y_i) \wedge \bigwedge \pi_x(C, y_i) \wedge \bigwedge y_i \not\approx y_j \\
\hline
\multicolumn{3}{c}{\text{Mapping Axioms to FOL}} \\
\hline
\pi(C \sqsubseteq D) & = & \forall x : \pi_y(C, x) \rightarrow \pi_y(D, x) \\
\pi(C \equiv D) & = & \forall x : \pi_y(C, x) \leftrightarrow \pi_y(D, x) \\
\pi(R \sqsubseteq S) & = & \forall x, y : R(x, y) \rightarrow S(x, y) \\
\pi(\mathsf{Trans}(R)) & = & \forall x, y, z : R(x, y) \wedge R(y, z) \rightarrow R(x, z) \\
\pi(C(a)) & = & \pi_y(C, a) \\
\pi((\neg)R(a, b)) & = & (\neg)R(a, b) \\
\pi(a \circ b) & = & a \circ b \text{ for } \circ \in \{\approx, \not\approx\} \\
\hline
\multicolumn{3}{c}{\text{Mapping } KB \text{ to FOL}} \\
\hline
\pi(R) & = & \forall x, y : R(x, y) \leftrightarrow R^-(y, x) \\
\pi(KB_\mathcal{R}) & = & \bigwedge_{\alpha \in KB_\mathcal{R}} \pi(\alpha) \wedge \bigwedge_{R \in N_{R_a}} \pi(R) \\
\pi(KB_\mathcal{T}) & = & \bigwedge_{\alpha \in KB_\mathcal{T}} \pi(\alpha) \\
\pi(KB_\mathcal{A}) & = & \bigwedge_{\alpha \in KB_\mathcal{A}} \pi(\alpha) \\
\pi(KB) & = & \pi(KB_\mathcal{R}) \wedge \pi(KB_\mathcal{T}) \wedge \pi(KB_\mathcal{A}) \\
\hline
\end{array}
$$

Notes:

(i): X is a meta variable and is substituted by the actual variable,

(ii): π_x is obtained from π_y by simultaneously substituting in the definition all $y_{(i)}$ with $x_{(i)}$, and by replacing π_y with π_x and vice versa.

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

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

select one representative role, denoted as $\mathsf{role}(\Gamma)$, such that, if $R \in \Gamma$, $\mathsf{Inv}(R) \in \Gamma'$ 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 Γ is the maximal, strongly connected component that R belongs to. Finally, we construct the new RBox $KB'_{\mathcal{R}}$ as follows: (i) 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 $KB'_{\mathcal{R}}$, and (ii) for each strongly connected component C, 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 also $\mathsf{Inv}(R) \in \Gamma$. 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 RBoxes are acyclic.

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

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

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:

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

Intuitively, $|KB|$ is the number of symbols needed to encode KB on the input tape of a Turing machine using the unary encoding of numbers. 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 unary coding of numbers: a number n can be encoded in unary coding with n bits.

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

- $\alpha|_\epsilon = \alpha$ for α 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 $\alpha|_3 = b$;
- $(a_1 \circ a_2)|_i = a_i$ for $\circ \in \{\approx, \not\approx\}$ and $i \in \{1,2\}$.

For α a $\mathcal{SHIQ}$ concept or an axiom and p a position in α, replacement of $\alpha|_p$ with β is denoted as $\alpha[\beta]_p$ and is defined in the obvious way (we assume that such replacements produce syntactically correct terms). 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 α into first-order logic, in the following way:

- $\mathsf{pol}(C, \epsilon) = 1$;
- $\mathsf{pol}(\neg C, 1.p) = -\mathsf{pol}(C, p)$;
- $\mathsf{pol}(C_1 \circ C_2, i.p) = \mathsf{pol}(C_i, p)$ for $\circ \in \{\sqcap, \sqcup\}$ and $i \in \{1,2\}$;
- $\mathsf{pol}(\bowtie R.C, 2.p) = \mathsf{pol}(C, p)$ for $\bowtie \in \{\exists, \forall\}$;
- $\mathsf{pol}(\leq n\, R.C, 3.p) = -\mathsf{pol}(C, p)$;
- $\mathsf{pol}(\geq n\, R.C, 3.p) = \mathsf{pol}(C, p)$;
- $\mathsf{pol}(C_1 \sqsubseteq C_2, 1.p) = -\mathsf{pol}(C_1, p)$ and $\mathsf{pol}(C_1 \sqsubseteq C_2, 2.p) = \mathsf{pol}(C_2, p)$;
- $\mathsf{pol}(C_1 \equiv C_2, i.p) = 0$ for $i \in \{1,2\}$;
- $\mathsf{pol}(C(a), 1.p) = \mathsf{pol}(C, p)$.

3.2 Description Logics with Concrete Domains

Practical applications of description logics usually require representing *concrete* properties such as height, name or age, having values from a fixed domain such as integers or strings, with *built-in* predicates. These requirements led to the extension of description logics with *concrete domains* [5]. Informally, a concrete domain consists of 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 [58] it was shown that a logic with general inclusion axioms and concrete domains is undecidable. Therefore, in [43, 70, 38] several alternatives have been investigated. The cumulative results of this research influenced the design of the Ontology Web Language (OWL) [71], which supports a basic form of concrete domains, so-called *datatypes*.

3.2.1 Concrete Domains

To present our definitions in a concise way, we introduce the following notation: with $\mathbf{x}$ we denote a vector of variables $x_1, \ldots, x_n$, and for a function δ, with $\delta(\mathbf{x})$ we denote the application of δ to each element x_i of $\mathbf{x}$, i.e. $\delta(x_1), \ldots, \delta(x_n)$.

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:*

- $\Phi_{\mathbf{D}}$ *is closed under negation, i.e. for each* $d \in \Phi_{\mathbf{D}}$*, there exists* $\overline{d} \in \Phi_{\mathbf{D}}$ *with* $\overline{d}^{\mathbf{D}} = \triangle_{\mathbf{D}}^{n} \setminus d^{\mathbf{D}}$*,*

- *it contains a unary predicate* $\top_{\mathbf{D}}$ *interpreted as* $\triangle_{\mathbf{D}}$*, and*

- $\mathbf{D}$*-satisfiability of finite conjunctions of the form* $\bigwedge_{i=1}^{n} d_i(\mathbf{x_i})$ *is decidable. The latter is the case if an assignment* δ *of variables to elements of* $\triangle_{\mathbf{D}}$ *exists, such that* $\delta(\mathbf{x_i}) \in d_i^{\mathbf{D}}$*, for each* $1 \leq i \leq n$*.*

Sometimes, we consider conjunctions over literals containing terms, rather than variables. Let $S = \{d_i(\mathbf{t_i})\}$ be a set of literals, where $\mathbf{t_i}$ is a vector of terms $t_{i1}, \ldots, t_{ik}$. With $\widehat{S}$ we denote 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.

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 Σ contains the sort c for the concrete objects, and the sort a for all other so-called *abstract* objects. When there is a need to distinguish the sorts syntactically, we denote the variables (function symbols) of sort c as x^{c} (f^{c}). Furthermore, since $\mathbf{D}$ does not provide an interpretation for concrete functions, we prohibit nesting of concrete terms.

Definition 3.2.2. *Let* $\mathbf{D}$ *be an admissible concrete domain, and* Σ *a signature such that* $\mathsf{c} \in \mathcal{S}$*,* $\Phi_{\mathbf{D}} \subseteq \mathcal{P}$*, the signature of each predicate from* $\Phi_{\mathbf{D}}$ *is* $\mathsf{c} \times \ldots \times \mathsf{c}$*, and, for each function symbol* f *with a signature* $r_1 \times \ldots \times r_n \rightarrow \mathsf{c}$*, we have* $r_i \neq \mathsf{c}$ *for each* i*. An interpretation* I *over the signature* Σ *is a* $\mathbf{D}$-interpretation *if (*i*)* $\mathcal{D}_{\mathsf{c}} = \triangle_{\mathbf{D}}$*, (*ii*) 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 in the standard way.*

When ambiguity does not arise, we do not stress $\mathbf{D}$ for satisfiability, equisatisfiability, entailment etc. We often assume that $\mathbf{D}$ contains a special equality predicate $\approx_{\mathbf{D}}$; such a predicate then has the interpretation $\{(x,x) \mid x \in \triangle_{\mathbf{D}}\}$. Since $\mathbf{D}$ is assumed to be admissible, the inequality predicate $\not\approx_{\mathbf{D}}$ is also in $\mathbf{D}$, and has the interpretation $\triangle_{\mathbf{D}} \times \triangle_{\mathbf{D}} \setminus \{(x,x) \mid x \in \triangle_{\mathbf{D}}\}$.

Usually, practical applications require several different concrete domains at once, such as strings and integers. In [5] an approach for combining two or more concrete domains into one has been presented. In this way, without losing generality we can assume only one concrete domain.

Definition 3.2.2 seems to have a drawback that it does not allow for explicit addressing of elements from the concrete domain. For example, it seems impossible to write $hasAge(peter, 15)$. However, one can always write $hasAge(peter, a_{15}) \wedge {=_{15}}(a_{15})$, where $=_{15}$ is a special concrete domain predicate with the singleton extension $\{15\}$. In the rest, we shall allow using individuals from the concrete domain in the formulae, while keeping in mind that the semantics is given by the above transformation.

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

We now present the formal definition of the $\mathcal{SHIQ}(\mathbf{D})$ description logic, which is obtained by combining $\mathcal{SHIQ}$ with an admissible concrete domain $\mathbf{D}$. The syntax

Table 3.3: Semantics of $\mathcal{SHIQ}(\mathbf{D})$ by Mapping to FOL

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

of $\mathcal{SHIQ}$ from Definition 3.1.1 can be extended to allow for concrete domains in the following way:

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 $\exists T_1, \ldots, T_m.d$, $\forall T_1, \ldots, T_m.d$, $\leq n\, T$, $\geq n\, T$, 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 individual names. *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 an abstract individual, and a^{c} and b^{c} are concrete individuals.*

A $\mathcal{SHIQ}(\mathbf{D})$ knowledge base is defined as in Definition 3.1.1, requiring that 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 can be extended to give semantics to $\mathcal{SHIQ}(\mathbf{D})$ in the following way:

Definition 3.2.4. *The semantics of a $\mathcal{SHIQ}(\mathbf{D})$ knowledge base KB is given by extending the mapping π from Definition 3.1.2 to translate KB into a multi-sorted first-order formula, as presented in Table 3.3. We assume the existence of a separate sort* a *for abstract objects, which is different from* c. *Atomic concept predicates have the signature* 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.*

Direct model-theoretic semantics can be easily extended to $\mathcal{SHIQ}(\mathbf{D})$, by assigning to each concrete individual a^{c} an element $(a^{\mathsf{c}})^I \in \triangle_{\mathbf{D}}$, and by assigning to each concrete

[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. Notice that this makes all concrete roles simple.

Table 3.4: 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		
$T \sqsubseteq U$	$T^I \subseteq U^I$	
$T(a,b^{\mathsf{c}})$	$(a^I,(b^{\mathsf{c}})^I)\in T^I$	
$\neg T(a,b^{\mathsf{c}})$	$(a^I,(b^{\mathsf{c}})^I)\notin T^I$	
$a^{\mathsf{c}} \approx b^{\mathsf{c}}$	$(a^{\mathsf{c}})^I = (b^{\mathsf{c}})^I$	
$a^{\mathsf{c}} \not\approx b^{\mathsf{c}}$	$(a^{\mathsf{c}})^I \neq (b^{\mathsf{c}})^I$	

role T a set $T^I \subseteq \triangle^I \times \triangle_{\mathbf{D}}$. The semantics of new concepts and axioms is given in Table 3.4, where $T_{(i)}$ and U are concrete roles, and d is a concrete predicate.

The rules for rewriting $\mathcal{SHIQ}(\mathbf{D})$ concepts into negation-normal form are given below:

$$\neg(\geq (n+1)\,T) \rightsquigarrow\ \leq n\,T \qquad \neg(\leq n\,T) \rightsquigarrow\ \geq (n+1)\,T$$
$$\neg(\exists T_1,\ldots,T_n.d) \rightsquigarrow \forall T_1,\ldots,T_n.\overline{d} \qquad \neg(\forall T_1,\ldots,T_n.d) \rightsquigarrow \exists T_1,\ldots,T_n.\overline{d}$$

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

- $|\exists T_1,\ldots,T_m.d| = |\forall T_1,\ldots,T_m.d| = 2+m$,
- $|\geq n\,T| = |\leq n\,T| = n+2$.

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

- 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.

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

For a $\mathcal{SHIQ}$ knowledge base KB, our ultimate goal is to derive a disjunctive datalog program $\mathsf{DD}(KB)$ which is satisfiable if and only if KB is. The intuitive principle by which the equisatisfiability of KB and $\mathsf{DD}(KB)$ is achieved is relatively simple. Namely, if KB is unsatisfiable, this 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 in $\mathsf{DD}(KB)$, 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 the simulation of inferences can be performed in both directions, $\mathsf{DD}(KB)$ and KB are equisatisfiable.

To obtain a sound, complete and terminating algorithm from the high-level idea outlined above, it is necessary to select the appropriate calculus $\mathcal{C}$, capable of effectively deciding satisfiability of KB. Positive disjunctive datalog is strongly related to clausal first-order logic. Intuitively, simulation of inferences using disjunctive datalog will be easier if $\mathcal{C}$ is a clausal refutation calculus. Hence, in the rest of this chapter we present a procedure for deciding satisfiability of $\mathcal{SHIQ}$ knowledge bases by basic superposition. The algorithm presented in this chapter is then used in Chapter 6 to obtain the reduction.

In Section 4.5 we give an overview of existing decision procedures for various decidable fragments of first-order logic based on clausal calculi. However, $\mathcal{SHIQ}$ cannot be directly embedded into any of these fragments. An exception is the two-variable fragment with counting quantifiers, but we find it difficult to use the decision procedure from [74] to simulate inference steps as sketched above. 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, it makes the logic lose the finite model property, and on the proof-theoretic side, tableau decision procedures for such logics require sophisticated pair-wise blocking techniques to ensure termination [44]. It turns out that this combination makes a resolution-based decision procedure more complicated as well: contrary to most existing decision procedures for related logics (such as [31] or [81]), to decide $\mathcal{SHIQ}$ it is necessary to consider clauses containing terms of depth two. To block certain undesirable inferences with such clauses, a stronger calculus for equality than superposition [9] is needed. The solution we adopt is to use *basic superposition* [14, 65]

4.1 Decision Procedure Overview

Before delving into the details, in this section we present a high-level overview of the principles we base our decision procedure upon. The fundamental principles for deciding a first-order fragment $\mathcal{L}$ by resolution are known from [49]. First, one selects a sound and complete resolution calculus $\mathcal{C}$. Next, one identifies the set of clauses $\mathcal{N}_\mathcal{L}$ such that for a finite signature, $\mathcal{N}_\mathcal{L}$ is finite and each formula $\varphi \in \mathcal{L}$, when translated into clauses, produces clauses from $\mathcal{N}_\mathcal{L}$. Finally, one demonstrates *closure* of $\mathcal{N}_\mathcal{L}$ under $\mathcal{C}$, namely, 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 $\mathcal{N}_\mathcal{L}$.

Basic superposition alone unfortunately does not decide $\mathcal{SHIQ}$. A minor problem are transitivity axioms, which in a clausal representation require clauses without so-called *covering literals* — literals containing all variables of a clause [45]. As shown in [51], termination of resolution on such clauses is very difficult to achieve. To address this, in Section 4.2 we show 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 focus on deciding satisfiability of $\mathcal{ALCHIQ}$ knowledge bases only.

A significantly more complex problem is that basic superposition alone decides only $\mathcal{ALCHIQ}^-$, where number restrictions are allowed only on roles not having subroles. Namely, the combination of role hierarchies, inverse roles and counting quantifiers may produce clauses which, during saturation by basic superposition, produce terms of ever increasing depth, so the saturation does not necessarily terminate. We address this problem in two stages.

In Section 4.3 we present a decision procedure for deciding satisfiability of an $\mathcal{ALCHIQ}^-$ knowledge base KB. First, we preprocess KB into a clausal representation as explained in Subsection 4.3.1. Let us denote the resulting set of closures with $\Xi(KB)$. It is not difficult to see that $\Xi(KB)$ contains only closures of a syntactic form as in Table 4.1. 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 in Definition 4.3.3. 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 of [49]:

- We generalize the types of closures from Table 4.1 to so called $\mathcal{ALCHIQ}^-$-closures, which are presented in Table 4.2. By Lemma 4.3.4, we show that each closure occurring in $\Xi(KB)$ is an $\mathcal{ALCHIQ}^-$-closure.
- By Lemma 4.3.5 we show that in any $\mathcal{BS}_{DL}$-derivation starting from $\Xi(KB)$, each inference produces either an $\mathcal{ALCHIQ}^-$-closure, or a closure which is redundant (and may be deleted).
- By Lemma 4.3.8 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 now a simple consequence of these two lemmata: in the worst case, one will derive all possible $\mathcal{ALCHIQ}^-$-closures, after which all further inferences are redundant. The bound on the size of the set of $\mathcal{ALCHIQ}^-$-closures gives us the complexity of the decision procedure, as demonstrated by Theorem 4.3.9.

To handle $\mathcal{ALCHIQ}$, in Subsection 4.4.1 we extend the basic superposition calculus with a new *decomposition* inference rule. This rule transforms certain closures with undesirable terms into several simpler closures. We show that decomposition is sound and complete. Furthermore, in in Subsection 4.4.2 we show that it guarantees the termination of basic superposition for $\mathcal{ALCHIQ}$. It turns out that the decomposition rule is quite general and versatile; in Subsection 4.4.3 we use it to decide a slightly stronger logic $\mathcal{ALCHIQ}b$, which allows certain *safe* role expressions.

4.2 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 $\mathcal{ALCHIQ}$ knowledge base $\Omega(KB)$. Since $\Omega(KB)$ is satisfiable exactly when KB is, without loss of generality we restrict our attention in the remaining sections to $\mathcal{ALCHIQ}$ knowledge bases. As one can easily see from Definition 4.2.2, for a $\mathcal{SHIQ}^-$ knowledge base KB, $\Omega(KB)$ is an $\mathcal{ALCHIQ}^-$ knowledge base. Therefore, we do not consider very simple roles explicitly in the definition of Ω.

The transformation presented here is similar to the one found in [86], where an algorithm for transforming $\mathcal{SHIQ}$ concepts to concepts in $\mathcal{ALCIQ}b$ logic was presented ($\mathcal{ALCIQ}b$ does not provide role hierarchy, but allows certain types of boolean operations on roles). Another similar transformation has been presented in [81], where it is demonstrated, among others, how $\mathrm{K4}_m$ formulae (i.e. the formulae of multi-modal logic with transitive modalities) can be encoded into K_m formulae (i.e. the formulae of multi-modal logic without transitive modalities).

Definition 4.2.1. *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 occurs in 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 $\forall S.C \in \mathsf{clos}(KB)$.*

Notice that all concepts in $\mathsf{clos}(KB)$ are in NNF. We define next the operator Ω which encodes any $\mathcal{SHIQ}$ knowledge base KB into an equisatisfiable $\mathcal{ALCHIQ}$ knowledge base $\Omega(KB)$.

Definition 4.2.2. *For a $\mathcal{SHIQ}$ knowledge base KB, let $\Omega(KB)$ denote the following $\mathcal{ALCHIQ}$ knowledge base:*

- *$\Omega(KB)_{\mathcal{R}}$ 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 $\mathsf{Trans}(S) \in KB_{\mathcal{R}}$,*
- *$\Omega(KB)_{\mathcal{A}} = KB_{\mathcal{A}}$.*

Observe that, for any concept C, the number of subconcepts in $\mathsf{clos}(KB)$ is bounded by the number of subexpressions of C. Furthermore, for each concept from $\mathsf{clos}(KB)$, we generate at most $|N_R|$ axioms in $\Omega(KB)_{\mathcal{T}}$. Hence, the encoding is polynomial in $|KB|$. Furthermore, it does not affect satisfiability, as we show next.

Theorem 4.2.3. *KB is satisfiable if and only if $\Omega(KB)$ is satisfiable.*

Proof. ($\Rightarrow$) Assume that I is a model of KB, but does satisfy an axiom of $\Omega(KB)$. 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 4.2.2. Hence, there is a domain element α 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 β 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 β such that $(\alpha, \beta) \in S^I$. There are two possibilities:
  - If no domain element γ exists such that $(\beta, \gamma) \in S^I$, then $\beta \in (\forall S.C)^I$.
  - If there is γ 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 γ, we have $\beta \in (\forall S.C)^I$.

  Either way, for any β, 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 may focus only on 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 $\mathsf{Trans}(R) \in KB_{\mathcal{R}}$, $R^{I'} = (R^I)^+$.
- If $\mathsf{Trans}(R) \notin KB_{\mathcal{R}}$, $R^{I'} = R^I \cup \bigcup_{S \sqsubseteq^* R, S \neq R} S^{I'}$.

Since we may assume that $KB_{\mathcal{R}}$ is acyclic, the above induction is well-defined. We will now show that I' satisfies every RBox axiom of KB. 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 from the fact that $A^+ \cup B^+ \subseteq (A \cup B)^+$ for any A and B. Furthermore, for any role R we have $R^I \subseteq R^{I'}$, and, 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 $\mathsf{NNF}(\neg C)$ occurs in D. We now show by induction on $\lessdot$ that, for each $D \in \mathsf{clos}(KB)$, $D^I \subseteq D^{I'}$. The relation $\lessdot$ is obviously acyclic, so the induction is well-founded. For the base case when 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:

- $D = C_1 \sqcap C_2$. Assume for some α we have $\alpha \in (C_1 \sqcap C_2)^I$. 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 $\alpha \in (C_1 \sqcap C_2)^{I'}$.
- $D = C_1 \sqcup C_2$. Assume for some α we have $\alpha \in (C_1 \sqcup C_2)^I$. Then either $\alpha \in C_1^I$, so by the induction hypothesis $\alpha \in C_1^{I'}$, or $\alpha \in C_2^I$, so by the induction hypothesis $\alpha \in C_2^{I'}$. Either way, $\alpha \in (C_1 \sqcup C_2)^{I'}$.

- $D = \exists R.C$. Assume for some α we have $\alpha \in (\exists R.C)^I$. Then there is a β 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 $\alpha \in (\exists R.C)^{I'}$.

- $D = \forall R.C$. Assume that $\alpha \in (\forall R.C)^I$. If there is no object β such that $(\alpha, \beta) \in R^{I'}$, then $\alpha \in (\forall R.C)^{I'}$. Otherwise, assume there is such a β. 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 there must be a role $S \sqsubseteq^* R$ with $\mathsf{Trans}(S) \in KB_{\mathcal{R}}$, and a path $(\alpha, \gamma_1) \in S^I$, $(\gamma_1, \gamma_2) \in S^I$, ..., $(\gamma_{n-1}, \beta) \in S^I$, $n > 1$. 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 $1 < i < n$. For $n - 1$ we have $\beta \in C^I$, so by the induction hypothesis $\beta \in C^{I'}$.

  Since for any β, in both cases we have that $\beta \in C^{I'}$, we have $\alpha \in (\forall R.C)^{I'}$.

- $D = \geq n\, R.C$. Assume that $\alpha \in (\geq n\, R.C)^I$. Then there are at least n distinct domain elements β_i such that $(\alpha, \beta_i) \in R^I$ and $\beta_i \in C^I$, so by the induction hypothesis $\beta_i \in C^{I'}$. Since $R^I \subseteq R^{I'}$, we have $\alpha \in (\geq n\, R.C)^{I'}$.

- $D = \leq n\, R.C$. Since R is simple, $R^I = R^{I'}$. 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 a β must exist 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, $\alpha \in (\leq n\, R.C)^{I'}$.

Now it is obvious that any 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 $a^{I'} \in C^{I'}$. 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 $C \equiv D$. □

Notice that for α of the form $(\neg)A(a)$ with A an atomic concept, or of the form $(\neg)R(a, b)$ with R 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 thus obtained 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 such ground queries where R is a complex role. This is why we restrict negative ground role atoms in Definition 3.1.1 of $\mathcal{SHIQ}$ knowledge bases to simple roles only.

4.3 Deciding $\mathcal{ALCHIQ}^-$

We now present in detail the algorithm for deciding satisfiability of an $\mathcal{ALCHIQ}^-$ knowledge base KB.

4.3.1 Preprocessing

To decide satisfiability of KB, we need to transform it into a clausal form. Straightforward transformation of $\pi(KB)$ into disjunctive normal form has two significant drawbacks. Firstly, the structure of formulae would be destroyed. Secondly, the usual transformation into clausal normal form might increase the size of the closure set exponentially. To overcome these drawbacks, we first apply the *structural transformation* [73, 68, 8], also known as *renaming*. Intuitively, for some first-order formula φ, the structural transformation introduces a new name for each subformula of φ. Thus, the original formula structure is preserved and, since the number of subformulae of φ is linear in the size of φ, the exponential blowup is avoided.

In the rest, we assume without loss of generality that all ABox concept membership axioms in KB are expressed using atomic concepts: for each membership axiom $C(a)$ where C is not atomic, one may 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, and we call such knowledge bases *extensionally reduced*.

Definition 4.3.1. *Let C be a concept and Λ 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:*

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

where Q is a new globally unique atomic concept. Furthermore, let

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

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

Notice that, for $C = D \sqcup \bigsqcup_i (\neg)A_i$ with A_i atomic concepts and D a complex concept, one can omit the position 1 in $\Lambda(C)$, since this reduces the number of closures generated. For example, the negation normal form of the axiom $\neg C \sqcap \neg D \sqsubseteq \exists R.\top$ is $C \sqcup D \sqcup \exists R.\top$, which can be readily transformed into a closure $C(x) \vee D(x) \vee R(x, f(x))$, without introducing a new name for the subconcept $\exists R.\top$. This optimization is, however, not essential for the correctness of the algorithm, so we do not make it a part of the definition.

Lemma 4.3.2. *Let KB be an $\mathcal{ALCHIQ}$ knowledge base. Then KB is satisfiable if and only if $\Xi(KB)$ is satisfiable. Furthermore, $\Xi(KB)$ can be computed in time polynomial in $|KB|$ for unary coding of numbers in input.*

Table 4.1: 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	$\bigvee(\neg)C_i(x) \vee \bigvee_{i=1}^{n} \neg R(x,y_i) \vee \bigvee_{i=1}^{n} D(y_i) \vee \bigvee_{i,j=1;j>i}^{n} y_i \approx y_j$
8	$(\neg)C(a)$
9	$(\neg)R(a,b)$
10	$a \approx b$
11	$a \not\approx b$

Proof. Let $\psi_C = \bigwedge_{D \in \mathsf{Def}(C)} \forall x : \pi_y(D,x)$. It is easy to see that ψ_C is actually the definitional normal form of $\varphi = \forall x : \pi_y(C,x)$ with respect to the set of positions of all non-atomic subformulae of φ. By the definition of π and Cls, and since transformation into definitional normal form does not affect satisfiability, it is easy to see that KB and $\Xi(KB)$ are equisatisfiable. 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 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 the skolemization of $\forall x : \pi_y(D,x)$ 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 obviously be computed in time polynomial in $|D|$, thus implying the claim of the lemma. □

Using binary coding, a number n can be represented with $\lceil \log_2 n \rceil$ bits. In this case, the number of function symbols introduced by skolemization is exponential in the size of the input, so translation into first-order logic would incur exponential blowup.

By definition of π from Table 3.1, it is easy to see that all closures obtained by this transformation share some common syntactic properties. Table 4.1 lists the types of closures that $\Xi(KB)$ may contain.

4.3.2 Parameters for Basic Superposition

For the following definition it is necessary to keep in mind that literals $A(t_1,\ldots,t_n)$ are encoded as $A(t_1,\ldots,t_n) \approx \top$, as discussed in Section 2.4. Due to this encoding, predicate symbols become function symbols, and atoms become $\mathcal{E}$-terms.

Definition 4.3.3. *Let $\mathcal{BS}_{DL}$ denote the $\mathcal{BS}$ calculus parameterized as follows:*

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

In $\mathcal{BS}_{DL}$, we need to compare $\mathcal{E}$-terms only in closures of types 3 – 6 and 8 from Table 4.2. It is easy to see that, since LPOs are total on ground $\mathcal{E}$-terms, and $\mathcal{E}$-terms in closures of type 3 – 6 and 8 have at most one variable, any LPO is total on non-ground $\mathcal{E}$-terms from these closures. In this case, one can use a more direct definition of the literal ordering. We associate with each literal $L = s \circ t$, $\circ \in \{\approx, \not\approx\}$, the triple $c_L = (\max(s,t), p_L, \min(s,t))$, where $\max(s,t)$ is the bigger of the two $\mathcal{E}$-terms, p_L is 1 if $\circ$ is $\not\approx$, and 0 otherwise, and $\min(s,t)$ 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. An LPO is used to compare the first and the third position of c_L, where for the second position we take $1 \succ 0$. It is easy to see that, since $\succ$ is total on $\mathcal{E}$-terms, this definition is equivalent to the one based on the two-fold multiset extension, given in Section 2.4.

Ordering and selection constraints in $\mathcal{BS}$ are checked *a posteriori*, that is, after computing the unifier. This is more general, since some $\mathcal{E}$-terms may be comparable only after unification. For example, $s = f(x)$ and $t = y$ are not comparable using an LPO. However, for $\sigma = \{x \mapsto a, y \mapsto g(f(a))\}$, we have $t\sigma \succ s\sigma$. The drawback 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 may 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 any substitution σ, since LPOs are stable under substitutions.

4.3.3 Closure of $\mathcal{ALCHIQ}^-$-closures under Inferences

We now generalize the types of closures from Table 4.1 to so-called $\mathcal{ALCHIQ}^-$-closures 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 $\mathbf{P_1}(f_1(x)) \vee \ldots \vee \mathbf{P_m}(f_m(x))$. Notice 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, some of the disjuncts may be empty. Finally, with $\approx / \not\approx$ we denote a positive or a negative equality literal.

Lemma 4.3.4. *Each closure from $\Xi(KB)$ is of exactly one of the types from Table 4.2. 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 $\mathbf{P^f}(x)$ is called the f-*support*, and R is called the* designated role *for f and is denoted as* role(f).

Proof. The first claim follows trivially from the definition of $\Xi(KB)$. 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 will be associated with exactly one closure of type 3. □

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

Lemma 4.3.5. *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.*

Table 4.2: 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	$\mathbf{P_1}(x) \vee \mathbf{P_2}([g(x)]) \vee \mathbf{P_3}(\langle \mathbf{f}([g(x)]) \rangle) \vee \bigvee \langle t_i \rangle \approx / \not\approx \langle t_j \rangle$ where t_i and t_j are either of the form $f([g(x)])$ or of the form x
7	$\mathbf{P_1}(x) \vee \bigvee \neg R(x, y_i) \vee \mathbf{P_2}(\mathbf{y}) \vee \bigvee y_i \approx y_j$
8	$\mathbf{R}(\langle \mathbf{a} \rangle, \langle \mathbf{b} \rangle) \vee \mathbf{P}(\langle \mathbf{t} \rangle) \vee \bigvee \langle t_i \rangle \approx / \not\approx \langle t_j \rangle$ where t, t_i and t_j are either some constant b or a functional 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 containing 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 $[f_i(a)] \approx [b]$ a *witness* closure of the form $R(\langle a \rangle, \langle b \rangle) \vee D$ exists, with $\mathsf{role}(f_i) = R$, D does not contain functional terms or negative binary literals, and is contained in this closure.

Proof. We first prove the property (max), determining which literals can be maximal in closures of types 3, 4, 5, 6 and 8 under ordering and selection function of $\mathcal{BS}_{DL}$:

- 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, the literal of the form $(\neg)P(x)$ can be maximal only if the closure does not contain a term of the form $f(x)$.
- In a closure of type 6, only a literal containing a term of the form $f([g(x)])$ can be maximal.
- In a closure of type 8, a literal of the form $(\neg)R(a,b)$, $(\neg)P(a)$, $a \approx b$, or $a \not\approx b$ can be maximal only if the closure does not contain a function symbol.

For closures of type 3 and 4, the claims follow directly from the properties of the precedence $>_P$ from Definition 4.3.3 and the definition of LPO. Furthermore, for any term t, function symbol f, and predicate symbol P, since $f >_P P$ and $f(t) \succ t$, we have $f(t) \succ P(t)$. For a closure of type 5, we have $P'(f(x)) \succ f(x) \succ P(x)$, so $P(x)$ can be maximal only if the closure does not contain a term of the form $f(x)$. For a closure of type 6, we have $P''(f(g(x))) \succ f(g(x)) \succ P'(g(x)) \succ g(x) \succ P(x)$, so only a literal containing a term of the form $f(g(x))$ may be maximal. Finally, for any function symbol f, constants a, b, and c, unary predicate symbol P and a binary predicate symbol R, by the properties of $>_P$ we have $f(a) \succ P(b)$, $f(a) \succ R(b,c)$ and

$f(a) \succ b$. Hence, a literal not containing function symbols can be maximal only if a closure it occurs in does not contain a function symbol.

We now prove the lemma by induction on the derivation length. By Lemma 4.3.4, 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 show first that the conclusion has the structure of an $\mathcal{ALCHIQ}^-$-closure, and later show 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 type 1 and 2 can participate only as negative 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 position. The following pairs of terms from premises can be unifiable:

- x and x'; $f(x)$ and $f(x')$; or $f([g(x)])$ and $f([g(x')])$. The unifier is $\sigma = \{x \mapsto x'\}$, and the conclusion is obviously a closure of type 5 or 6. Notice that superposition from $f(g(x)) \approx x$ into $f(g(x')) \not\approx x'$ results in $x' \not\approx x'$, which can eagerly be eliminated by reflexivity resolution.
- 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.
- 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 since a term of the form $f(g(x))$ does not unify with a term of the form a or $f(a)$. For closures of type 5 and 8, the unifier may be $\sigma = \{x \mapsto a\}$ or $\sigma = \{x \mapsto f(a)\}$, and the conclusion is of type 8. Inferences between closures of type 8 have an empty unifier and 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 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 may be assigned indices 1 and 2. Since $g_1(x_1)$ and $g_2(x_2)$ must be unified with x, σ 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 i and j, the conclusion contains $x_1 \approx x_1$, so it is a tautology.

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

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

- Assume some side premises have the maximal literal of the form $R(x_i, \langle f_i(x_i) \rangle)$ 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 since all these arguments should unify with x). The unifier σ 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 $k+1 \leq i \leq n$. The conclusion is of type 8.

Superposition into a closure of type 3. The only remaining possible inference is superposition into a closure of type 3 with the free variable x'. By condition (max), $(w \approx v) \cdot \rho$ 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 $[f(\iota)] \approx [g(\iota)]$. The unifier σ is $\{x' \mapsto \iota\}$, so the conclusion is $S = \mathbf{P^f}([\iota]) \vee R([\iota], [g(\iota)]) \vee C \cdot \rho$, where $\mathbf{P^g}([\iota]) \subseteq C \cdot \rho$. By Condition (iv), the R^g-generator $P^g(y) \vee R(y, g(y))$ exists, so it subsumes S by the substitution $\{y \mapsto \iota\}$.

- $(C \vee s \approx t) \cdot \rho$ is a closure of type 6 with $(s \approx t) \cdot \rho$ of the form $[f(g(x))] \approx x$. The unifier σ is $\{x' \mapsto g(x)\}$, 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, so 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 σ is $\{x' \mapsto a\}$, 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. If the witness has been subsumed by some other closure, then, since the subsumption relation is transitive, this other closure subsumes the conclusion.

In all cases, the superposition conclusion is subsumed by an existing closure, so a superposition into a closure of type 3 is always redundant.

Equality factoring. Ordering constraints allow optimizing 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 σ 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 $\top$. 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 we rewrite the ordering constraints as $t\rho \prec s\rho$ and $t'\rho \prec s'\rho$. By the fact that $s\rho \approx t\rho$ is strictly maximal, we conclude that $t\rho \succ t'\rho$.

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 type 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 $t\rho \succ t'\rho$. Thus, the conclusion is a basic tautology and is redundant.

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 $\langle a \rangle \approx \langle b \rangle$ with at least one non-marked term. Depending on the marking, equality factoring either yields a basic tautology which is redundant or a closure of type 8. Notice 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, i.e. the 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 subsumes the premise, so this inference should be applied eagerly.

Conditions. From the above case analysis, one may observe that the following property (uni) hold: closures of type 3, 4 and 5 with a free variable x participate in inferences only with a unifier containing mappings $x \mapsto x'$, $x \mapsto g(x')$, $x \mapsto f(g(x'))$ or $x \mapsto a$. We now show that all conditions from Table 4.2 hold for each non-redundant inference conclusion.

Condition (i): If a closure C satisfies Condition (i), by the property (uni) the unifier σ may only instantiate x. Hence, $C\sigma$ will satisfy (i) as well. Since no inference removes markers from functional terms, 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 the property (max) literals from $\mathbf{P^f}([t])$ cannot participate in an inference. Furthermore, by the property (uni), any variable x is instantiated simultaneously in $f([t])$ and $P^f([t])$. The functional terms in the conclusion always stem from one of its premises, so Condition (iii) holds for any conclusion.

Conditions (iv) and (v): All equality literals with functional terms are generated by hyperresolution with the nucleus C of type 7. Since the role R occurring in C is very simple, a closure of type 3 or 4 cannot be resolved with a closure of type 2. Hence, for all electrons of the form $\mathbf{P^f}(x) \vee R(x, \langle f(x) \rangle)$ we have $\mathsf{role}(f) = R$, and for an electron 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 equality literals with functional terms is not possible, and in any literal $[f_i(x)] \approx [f_j(x)]$ the variable x is instantiated simultaneously. Hence, Conditions (iv) and (v) hold for each conclusion of any inference.

Condition (vi): All literals of the form $[f(a)] \approx [b]$ are generated by hyperresolution involving an electron E_1 of type 8 with the maximal literal $R(\langle a \rangle, \langle b \rangle)$ and an electron E_2 of type $\mathbf{P^f}(x) \vee R(x, \langle f(x) \rangle)$. Since R occurring in such C must be 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 the property (max) no literal from E_1

contains a functional term, 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 no literal from D contains a functional term or is a negative binary literal, by the property (max) no literal from C occurring in D may participate in an inference, so all literals are present in the conclusion. Finally, both sides of the all equality literals containing functional terms are marked, so each equality literal with functional terms derived in an inference must always occur in some of the premises. Hence, Condition (vi) holds for each conclusion. □

The following corollary can be easily shown by inspecting the proof of Lemma 4.3.5:

Corollary 4.3.6. *If a closure of type 8 participates in a $\mathcal{BS}_{DL}$ inference in a derivation from Lemma 4.3.5, the unifier σ 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 certain algorithms we present later.

Corollary 4.3.7. *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 functional terms. Furthermore, a closure of type 8 can participate in an inference only with closures not containing functional terms.*

Proof. For a KB as in the corollary, with (*) we denote the fact that each closure of type 7 in $\Xi(KB)$ contains exactly one literal $\neg R(x, y)$ and does not contain equality literals. Now the claim of the corollary can be shown by induction on the derivation length. The base case is obvious, since all ABox closures in $\Xi(KB) = N_0$ are of type from Table 4.1, so they do not contain functional terms. 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 functional term might be generated only by a hyperresolution with a closure of type 7 containing equality literals, but this is not possible by (*). A literal $(\neg)C([g(a)])$ might be derived by hyperresolving a closure of type 7 with an electron 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 $(\neg)C(f(x))$, but this is not possible, since N_i does not contain equality literals with functional terms. Finally, since a closure of type 8 does not contain a functional term, it can participate in an inference only with a literal not containing a functional term. Since such a literal must be maximal, it may occur only in a closure not containing a functional term. □

4.3.4 Termination and Complexity Analysis

We show that the number of $\mathcal{ALCHIQ}^-$-closures is finite for a finite signature. This, in combination with Lemma 4.3.5 and the soundness and completeness of $\mathcal{BS}_{DL}$ shows that $\mathcal{BS}_{DL}$, with eager application of redundancy elimination rules, is a decision procedure for checking satisfiability of $\mathcal{ALCHIQ}^-$ knowledge bases.

Lemma 4.3.8. *Let N_i be any closure set obtained in a derivation as defined in Lemma 4.3.5. 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 4.3.5, N_i can contain only $\mathcal{ALCHIQ}^-$-closures. Since redundancy elimination is applied 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 $2r \cdot 2c \cdot 2c + 2a \cdot 2c + 2a \cdot 2f \cdot c + 2 \cdot (4c^2 + c \cdot f^2 + cf \cdot 2c)$: the factor 2 in front of the parenthesis takes into account that equalities and inequalities may have the same form, and the expression in the parenthesis counts all possible forms these literals may 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 steps 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 is polynomial in $|KB|$, for unary coding of numbers.

The maximal number of closures of type $1 - 6$ and 8 in N_i is now easily obtained as follows: if C_ℓ is the closure with maximal number of literals ℓ for some closure type, then there are 2^ℓ subsets of literals of C_ℓ. To obtain the total number of closures, one must multiply 2^ℓ 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 which are contained in $\Xi(KB)$. □

Theorem 4.3.9. *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 4.3.2, and contains only $\mathcal{ALCHIQ}^-$-closures by Lemma 4.3.4. Let ℓ denote the maximal number of closures occurring in the closure set in a derivation as specified in Lemma 4.3.5, and let l denote the maximal number of literals in a closure. By Lemma 4.3.8, ℓ is exponential, and l polynomial in $|KB|$, for unary coding of numbers. Since all terms are bounded in size, two terms can be compared in constant time, and a maximal term in a closure can be selected in time linear in l. In [33], a subsumption decision algorithm was presented, running in time exponential in l. Furthermore, the subsumption check is performed at most for each pair of closures, so it takes at most exponential time. Apart from ordered hyperresolution with a closure of type 7, each of the four inference rules can potentially be applied to any closure pair. Since for closures other than of type 7 exactly one literal is maximal/selected, this gives rise to at most $4\ell^2$ inferences. For a hyperresolution inference with a closure of type 7, n side premises can be chosen in ℓ^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, after at most an exponential number of steps, the set of closures will be saturated, and the procedure will terminate. 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 4.3.9, we assumed an exponential algorithm for checking subsumption. In practice, it is known that modern theorem provers spend up to 90% of their time in subsumption checking, so efficient subsumption checking is crucial for practical applicability of resolution theorem proving.

Fortunately, for $\mathcal{ALCHIQ}^-$-closures subsumption checks can be performed in polynomial time, as follows. In [33] it was shown that subsumption between closures having at most one variable can be checked in polynomial time. This algorithm can be easily extended to additionally check η-reducibility, so we get 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 σ 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 since the closures C, $\mathbf{P_1}(x)$ and $\mathbf{P_2}(y_i)$ contain only one variable.

4.4 Removing the Restriction to Very Simple Roles

In this section we show how to remove the restriction to very simple roles and thus obtain an algorithm for deciding satisfiability of an $\mathcal{ALCHIQ}$ knowledge base KB. Our main problem is that by saturating $\Xi(KB)$, we may obtain closures whose structure corresponds to the Table 4.2, but for which conditions (iii) – (vi) do not hold; we call such closures $\mathcal{ALCHIQ}$-closures.

To better understand the problems encountered by removing the restriction to very simple roles, consider the following example, where KB is a knowledge base containing axioms (4.1) – (4.9). On the right-hand side we show the translation of KB into closures $\Xi(KB)$:

$$
\begin{array}{llcr}
(4.1) & R \sqsubseteq T & & \neg R(x,y) \vee T(x,y) \\
(4.2) & S \sqsubseteq T & & \neg S(x,y) \vee T(x,y) \\
(4.3) & C \sqsubseteq \exists R.\top & & \neg C(x) \vee R(x, f(x)) \\
(4.4) & \top \sqsubseteq \exists S^{-}.\top & & S^{-}(x, g(x)) \\
(4.5) & \top \sqsubseteq \leq 1 T & \rightsquigarrow & \neg T(x,y_1) \vee \neg T(x,y_2) \vee y_1 \approx y_2 \\
(4.6) & \exists S.\top \sqsubseteq D & & \neg S(x,y) \vee D(x) \\
(4.7) & \exists R.\top \sqsubseteq \neg D & & \neg R(x,y) \vee \neg D(x) \\
(4.8) & \top \sqsubseteq C & & C(x) \\
(4.9) & & & \neg S^{-}(x,y) \vee S(y,x)
\end{array}
$$

Consider a saturation of $\Xi(KB)$ by $\mathcal{BS}_{DL}$. Resolving (4.4) with (4.9) yields (4.10). Furthermore, (4.3) and (4.10) are resolved with (4.1) and (4.2) to produce (4.11) and (4.12), respectively. These can then be resolved with (4.5) to produce (4.13):

$$
\begin{array}{lr}
(4.10) & S([g(x)]\,, x) \\
(4.11) & \neg C(x) \vee T(x, [f(x)]) \\
(4.12) & T([g(x)]\,, x) \\
(4.13) & \neg C([g(x)]) \vee [f(g(x))] \approx x
\end{array}
$$

For closure (4.13), condition (v) from Table 4.2 is not satisfied. Namely, we have $\mathsf{role}(f) = R \neq \mathsf{Inv}(\mathsf{role}(g)) = \mathsf{Inv}(S^{-}) = S$; this is because in (4.5) a number restriction was stated on a role that is not very simple. Now (4.13) can be superposed into (4.3), resulting in (4.14):

$$
(4.14) \qquad \neg C([g(x)]) \vee R([g(x)]\,, x)
$$

Since condition (v) is not satisfied for (4.13), we cannot assume that there is a closure that subsumes (4.14), as we did in the proof of Lemma 4.3.5. Hence, we must keep (4.14), which is obviously not an $\mathcal{ALCHIQ}$-closure. This might cause termination problems since, in general, (4.14) 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 the binary literal is now two, and it is not difficult to see that, by resolving (4.14) with some closure of type 7, it is possible to derive closures with ever deeper terms. Hence, Lemma 4.3.8, stating that the number of closures that can be derived is finite, does not hold any more, so saturation does not necessarily terminate.

A careful analysis of the problem reveals that various refinements of the ordering and the selection function will not help. Furthermore, (4.14) is necessary for completeness. Namely, KB is unsatisfiable, and the empty clause is derived by the following deduction, which involves (4.14):

(4.15) $$D([g(x)])$$
(4.16) $$\neg D([g(x)]) \vee \neg C([g(x)])$$
(4.17) $$\neg C([g(x)])$$
(4.18) $$\Box$$

4.4.1 Transformation by Decomposition

To remedy the problems outlined above, we introduce *decomposition* — an additional transformation which may be applied to the result of certain $\mathcal{BS}$ inferences. This transformation is generally applicable and is not limited to description logic. We show that decomposition can be combined with basic superposition, but in a similar way one can show that decomposition can be combined with any clausal calculus compatible with the general 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 4.4.1. *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: (*i*) $C = C_1 \cup C_2$, (*ii*) $\rho = \theta\{\mathbf{x} \mapsto \mathbf{t}\}$, (*iii*) $\mathbf{x}$ is exactly the set of free variables of $C_2\theta$, and (*iv*) 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 often written as*

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

Let ξ be a $\mathcal{BS}$ inference with a most general unifier σ on a literal $L_m \cdot \eta$ from a main premise $D_m \cdot \eta$ and with a side premise $D_s \cdot \eta$. The conclusion of ξ is eligible for decomposition *if, for each ground substitution τ 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 $C([g(x)]) \vee R([g(x)], [h(g(x))])$. The conclusion is obviously not an $\mathcal{ALCHIQ}$-closure, so performing further inferences with it might lead to non-termination. However, the closure can be decomposed into closures $C([g(x)]) \vee Q_{R,f}([g(x)])$ and $\neg Q_{R,f}(x) \vee R(x, [h(x)])$, which are both $\mathcal{ALCHIQ}$-closures, and do not cause termination problems. The inference is eligible for decomposition if we ensure that $Q_{R,f}([g(x)]) \prec R(g(x), f(g(x))) = L_m\eta\sigma$; this can be done by using a LPO with the precedence $>_P$ such that $R >_P Q_{R,f}$.

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

Lemma 4.4.2. *Let $N_0, \ldots, N_i$ be a $\mathcal{BS}^+$-derivation, and let I_0 be a model of N_0. Then for $i > 1$, N_i has a model I_i such that, if the inference deriving N_i from N_{i-1} involves a decomposition step as specified in Definition 4.4.1 introducing a new 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 $I_i = I_{i-1}$.*

Proof. The proof proceeds 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 trivial. For example, consider a positive superposition inference from $(s \approx t \vee C) \cdot \rho$ into $(w \approx v \vee D) \cdot \rho$ with unifier σ resulting in $(C \vee D \vee w[t]_p \approx v) \cdot \theta$, where $\theta = \rho\sigma$, and let τ 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 similar.

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} can be 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 any ground substitution τ, 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 4.4.3. *Let ξ be a $\mathcal{BS}$ inference applied to premises from a closure set N resulting in a closure $C \cdot \rho$. If $C \cdot \rho$ can be decomposed into closures $C_1 \cdot \rho \vee Q([\mathbf{t}])$ and $C_2 \cdot \theta \vee \neg Q(\mathbf{x})$ which are both redundant in N, then the inference ξ is redundant in N.*

Proof. Let ξ 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 σ resulting in a closure $C \cdot \rho$. Furthermore, let R be a rewrite system, and τ a ground substitution such that $\xi\tau$ satisfies the ordering constraints of $\mathcal{BS}$ and it is a variable irreducible ground instance of ξ 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, we have $D_s\eta\sigma\tau \prec L_m\sigma\eta\tau$. Furthermore, superposition inferences are allowed only from the maximal side of the equality, so the inference always produces a literal $L' \prec L_m\eta\sigma\tau$. Finally, by the assumption on eligibility of ξ for decomposition, $\neg Q(\mathbf{t})\tau \prec L_m\eta\sigma\tau$. This implies that, for each literal $L \succ L_m\eta\sigma\tau$, if L occurs in $E_1 \cup E_2$ n times, then L also occurs n times in D. In other words, both E_1 and E_2 contain at most those literals larger than $L_m\eta\sigma\tau$ which 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 τ such that $\xi\tau$ is a variable irreducible ground instance of ξ w.r.t. R, the closures E_1 and E_2 are $\prec$-smaller than D, and they are variable irreducible w.r.t R. The closure $C_1 \cdot \rho \vee Q([\mathbf{t}])$ is redundant in N by assumption, so $R \cup \mathsf{irred}_R(N)^{\preceq E_1} \models E_1$ 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 an complete calculus, as shown by Theorem 4.4.4. 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 that the term ordering is well-founded.

Theorem 4.4.4. *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 4.4.2. For the ($\Leftarrow$) direction, assume that N is saturated under $\mathcal{BS}^+$. Then, by Lemma 4.4.3, N is saturated under $\mathcal{BS}$ as well. Using the model generation method 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 may 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 τ, there are ground closures $D_i\tau \in \mathsf{irred}_R(N)$ such that $D_1\tau, \ldots, D_n\tau \models C\tau$. Hence, 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 η' be a substitution obtained from η by replacing each mapping $x \mapsto t$ with $x \mapsto \mathsf{nf}_R(t)$. Since the substitution part of C is empty, $C\eta' \in \mathsf{irred}_R(N_0)$ and, since $R^* \models C\eta'$, we have $R^* \models C\eta$. Hence, $R^* \models N_0$, and by Lemma 4.4.2, there is a model of N. $\square$

Discussion. We discuss the intuition behind the Theorem 4.4.4. 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 since, 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 "relevant" for detecting potential inconsistency of the closure set. Notice 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 which do not correspond to a variable irreducible ground instance of $C \cdot \rho$. However, these "excessive" variable irreducible ground instances do not cause problems, since decomposition is a sound inference.

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. For example, this might happen if the ordering constraints on the predicates are such that the obtained fixed and variable parts need to be resolved on literals $Q([\mathbf{t}])$ 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 make together the original inference redundant, so the inference does not need to be repeated in the theorem proving process.

Notion of Eligibility. We briefly discuss the notion of eligibility from Definition 4.4.1. It essentially ensures that the closures obtained by decomposition of the conclusion of an inference ξ are bigger than the main premise of ξ. Consider the following $\mathcal{BS}$ inference ξ followed by decomposition (the unifier is $\sigma = \{x' \mapsto x\}$):

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

To determine if ξ is eligible for decomposition, we have a problem that $\neg A(x)$, the main literal on which the inference takes place, 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 τ such that $\xi\tau$ satisfies the ordering constraints of $\mathcal{BS}$. For comparing terms, we shall assume a LPO as in Definition 4.3.3. Provided that $\neg A(x)$ is not selected, it must be that $\neg A(x)\sigma\tau \succ (B(x) \vee C(y))\sigma\tau$; this implies that $x\sigma\tau \succ y\sigma\tau$. If we ensure that $A >_P Q$ 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 bigger than $(B(x) \vee C(y))\sigma\tau$. Therefore, we cannot conclude that $x\sigma\tau \succ y\sigma\tau$, and that $\neg A(x)\sigma\tau \succ \neg Q(x,y)\sigma\tau$. In deed, it is possible to take $\tau = \{x \mapsto a, y \mapsto f(x)\}$; now $A(x)\sigma\tau \prec Q(x,y)\sigma\tau$, regardless of the LPO precedence. Hence, the eligibility condition is not satisfied.

By formulating the eligibility condition by referring to ground substitutions, we achieve a higher level of generality. However, in common cases decomposition is applied to closures of simpler syntactic structure, for which we derive simpler and sufficient eligibility tests.

Corollary 4.4.5. *Let ξ be a $\mathcal{BS}$ inference as in Definition 4.4.1. If $\neg Q(\mathbf{t}) \prec L_m\eta\sigma$, then ξ is eligible for superposition.*

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

Corollary 4.4.6. *Let ξ be a $\mathcal{BS}$ inference as in Definition 4.4.1. 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 ξ is eligible for superposition.*

Proof. For ξ a $\mathcal{BS}$ inference and τ a ground substitution as in Definition 4.4.1, it is well-known that $L_s\eta\sigma\tau \prec L_m\eta\sigma\tau$ for $L_s \cdot \eta$ 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 $\neg Q(\mathbf{t})\tau \prec L\eta\sigma\tau \preceq L_s\eta\sigma\tau \prec L_m\eta\sigma\tau$. □

Combining Decomposition with Other Calculi. For any other sound clausal calculus, Lemma 4.4.2 applies identically. Furthermore, for any calculus compatible with the standard notion of redundancy [13], Lemma 4.4.3 can be proved in a similar manner with minor differences.

4.4.2 Deciding $\mathcal{ALCHIQ}$ by Decomposition

We now extend the decision procedure from Section 4.3 with the decomposition rule from Subsection 4.4.1 to obtain a decision procedure for checking satisfiability of $\mathcal{ALCHIQ}$ knowledge bases.

Definition 4.4.7. *Let $\mathcal{BS}^+_{DL}$ be the $\mathcal{BS}_{DL}$ calculus where conclusions, whenever possible, are decomposed as follows, for an arbitrary term t:*

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

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

The precedence of the LPO is $f >_P c >_P p >_P Q_{S,f} >_P \top$, for any function symbol f, constant symbol c, non-definition predicate p and definition predicate $Q_{S,f}$.

By Definition 4.4.1, 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 4.4.1 would require introducing a distinct definition predicate $Q'_{R,f}$ for $R([f(x)], x)$. However, by the operator π for translating KB into first-order logic, $R([f(x)], x)$ and $\mathsf{Inv}(R)(x, [f(x)])$ are logically equivalent. Therefore, $Q_{\mathsf{Inv}(R),f}$ may 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 4.4.7. This optimization is not essential for our results; however, it is always a good practice to omit unnecessary predicate symbols if possible.

Intuitively, $\mathcal{BS}^+_{DL}$ decides satisfiability of $\mathcal{ALCHIQ}$ knowledge bases because decomposition replaces a non-$\mathcal{ALCHIQ}$-closure with two $\mathcal{ALCHIQ}$-closures. 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 may be introduced during saturation, so the result of Theorem 4.4.4 applies. Since the number of $\mathcal{ALCHIQ}$-closures is finite according to Lemma 4.3.8, $\mathcal{BS}^+_{DL}$ terminates.

Theorem 4.4.8. *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. The proof of Lemma 4.3.5 obviously applies even if conditions (iii) – (vi) from Table 4.2 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 π, 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.

Notice that $R(t, f(t)) = L_m \eta \sigma$ is the maximal literal of the main premise after applying the unifier and, since $R >_P Q_{R,g}$ in the LPO of $\mathcal{BS}^+_{DL}$, $\neg Q_{R,g}(t) \prec R(t, f(t))$, so by Corollary 4.4.5 the superposition inferences in the first two cases are eligible for decomposition. Notice also 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 Corollary 4.4.5), but this is not essential to obtain a decision procedure. To summarize, decomposition ensures that the conclusion of each inference of $\mathcal{BS}^+_{DL}$ is an $\mathcal{ALCHIQ}$-closure.

Let r be a number of roles and f the number of function symbols occurring in $\Xi(KB)$; as in Lemma 4.3.8, 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 then bounded by $r \cdot f$, which is quadratic in $|KB|$, so the number of different predicates is polynomial in $|KB|$. Hence, Lemma 4.3.8 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 4.4.4, the claim of this theorem follows. □

Although this result supersedes the Theorem 4.3.9, in practice it is useful to know that superposition into a 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.

Notice that Definition 4.4.7 applies decomposition aggressively. 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 4.4.7 all these conclusions should be decomposed, even though they are already $\mathcal{ALCHIQ}$-closures. We note that in these cases decomposition is optional: it may be performed, but is not strictly necessary to obtain termination.

For most knowledge bases, not all possible predicates $Q_{R,f}$ will be introduced in a saturation of $\Xi(KB)$. Rather, only predicates from the set $\mathsf{gen}(KB)$ defined below can be introduced by decomposition. This fact is used in the algorithm for reducing an $\mathcal{ALCHIQ}$ knowledge base KB to a disjunctive datalog program from Chapter 6.

Definition 4.4.9. *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:*

- *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 $\mathsf{Inv}(R) \sqsubseteq^* S$ and*
- *$\mathsf{role}(f) \sqsubseteq^* S$ or $\mathsf{Inv}(\mathsf{role}(f)) \sqsubseteq^* S$.*

Lemma 4.4.10. *In any saturation of $\Xi(KB)$ by $\mathcal{BS}^+_{DL}$, the decomposition rule introduces only definition closures from the set $\mathsf{gen}(KB)$.*

Proof. A predicate $Q_{R,f}$ is introduced by decomposing the conclusion of a superposition into a generator $\mathbf{P^g}(x) \vee R(x, g(x))$ from a literal of the form $[g(f(x))] \approx x$ where $\mathsf{role}(g) \neq \mathsf{Inv}(\mathsf{role}(f))$, or $[g(t)] \approx [f(t)]$ where $\mathsf{role}(g) \neq \mathsf{role}(f)$. Such a literal can only be generated by a resolution with a closure of type 7, obtained by translating an at-least restriction on some role S into clausal form. Unless $R \sqsubseteq^* S$ or $\mathsf{Inv}(R) \sqsubseteq^* S$, the function symbol g cannot occur in any literal of the form $[g(f(x))] \approx x$ or $[g(t)] \approx [f(t)]$. Similarly, unless $\mathsf{role}(f) \sqsubseteq^* S$ or $\mathsf{Inv}(\mathsf{role}(f)) \sqsubseteq^* S$, the function symbol f cannot occur in any such literal either. □

4.4.3 Safe Role Expressions

A prominent limitation of $\mathcal{ALCHIQ}$ is the rather restricted form of axioms regarding roles. These limitations may be partially overcome by allowing for *safe* Boolean role expressions in TBox and ABox axioms. The resulting logic is called $\mathcal{ALCHIQ}b$, and can be viewed as the "union" of $\mathcal{ALCHIQ}$ and $\mathcal{ALCIQ}b$ [86]. Roughly speaking, safe role expressions are built by combining simpler role expressions using union, disjunction, and *relativized* negation of roles. Thus "relativized" statements such as $\forall x, y : \mathit{isParentOf}(x, y) \rightarrow \mathit{isMotherOf}(x, y) \vee \mathit{isFatherOf}(x, y)$ are allowed, but "fully negated" statements such as $\forall x, y : \neg \mathit{isMotherOf}(x, y) \rightarrow \mathit{isFatherOf}(x, y)$ are not. The safety restriction is needed for the algorithm to remain in ExpTime; namely, it is known that reasoning with non-safe role expressions is NExpTime-complete [59].

Definition 4.4.11. *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* safe^+ *and* safe^- *be functions defined on the set of all role expressions as specified below, where* R *is an abstract role, and* E *and* E_i *are role expressions:*

$$\begin{aligned}
\mathsf{safe}^+(R) &= \mathit{true} & \mathsf{safe}^-(R) &= \mathit{false} \\
\mathsf{safe}^+(\neg E) &= \neg\mathsf{safe}^-(E) & \mathsf{safe}^-(\neg E) &= \neg\mathsf{safe}^+(E) \\
\mathsf{safe}^+(\sqcup E_i) &= \bigwedge \mathsf{safe}^+(E_i) & \mathsf{safe}^-(\sqcup E_i) &= \bigvee \mathsf{safe}^+(E_i) \\
\mathsf{safe}^+(\sqcap E_i) &= \bigvee \mathsf{safe}^+(E_i) & \mathsf{safe}^-(\sqcap E_i) &= \bigwedge \mathsf{safe}^+(E_i)
\end{aligned}$$

A role expression E *is* safe *if* $\mathsf{safe}^+(E) = \mathit{true}$*. 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* π *as specified in Table 4.3.*

In [86], a similar logic $\mathcal{ALCIQ}b$ was considered. There, the safety condition for role expressions required that, if the expression is transformed into disjunctive normal

Table 4.3: Semantics of Role Expressions

$$\begin{aligned}
\pi(R, X, Y) &= R(X, Y) \\
\pi(\neg R, X, Y) &= \neg R(X, Y) \\
\pi(\sqcap E_i, X, Y) &= \bigwedge E_i(X, Y) \\
\pi(\sqcup E_i, X, Y) &= \bigvee E_i(X, Y)
\end{aligned}$$

form, then each disjunct contains at least one non-negated conjunct. It is straightforward to see that these two definitions coincide. We use the above definition because transformation into disjunctive normal form can introduce exponential blowup, which we avoid by using the 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 non-atomic role expression. The set of closures obtained by preprocessing is also denoted with $\Xi(KB)$. From [68] it is known that $\Xi(KB)$ can be computed in polynomial time.

Theorem 4.4.12. *For an $\mathcal{ALCHIQ}b$ knowledge base KB, saturation of $\Xi(KB)$ by $\mathcal{BS}^{+}_{DL}$ decides satisfiability of KB in time 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 new predicate:

$$\forall x, y : Q(x, y) \rightarrow \pi(E, x, y) \tag{4.19}$$

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:

$$\forall x, y : \pi(E, x, y) \rightarrow Q(x, y) \tag{4.20}$$

Finally, for an inclusion axiom $E \sqsubseteq F$, the structural transformation introduces formulae of the form (4.19) and (4.20). Regardless of whether E is safe in (4.19) or not, the structural transformation and clausification of (4.19) produces closures containing the literal $\neg Q(x, y)$. Furthermore, in (4.20), E is safe, which means that, in the disjunctive normal form of E, each conjunct contains at least one positive literal. Hence, each disjunction 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 (4.20) also produces a closure with at least one such literal. Hence, all closures produced by role expressions have the form (4.21), where $n > 0$ and $m \geq 0$:

$$\neg R_1(x, y) \vee \ldots \vee \neg R_n(x, y) \vee S_1(x, y) \vee \ldots \vee S_m(x, y) \tag{4.21}$$

Since such a closure always contains at least one negative literal, it 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 $R([f(x)], x)$ (since $f(x)$ 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 the form $R_i(x_i, \langle f(x_i)\rangle)$, the resolvent has the form (4.22), for $\mathbf{S}(s,t) = S_1(s,t) \vee \ldots \vee S_m(s,t)$:

$$\mathbf{P}(x) \vee \mathbf{S}(x, [f(x)]) \tag{4.22}$$

This closure can be decomposed into (4.23) – (4.25):

$$\mathbf{P}(x) \vee Q_{S_1,f}(x) \vee \ldots \vee Q_{S_m,f}(x) \tag{4.23}$$
$$\neg Q_{S_1,f}(x) \vee S_1(x, [f(x)]) \tag{4.24}$$
$$\vdots$$
$$\neg Q_{S_m,f}(x) \vee S_m(x, [f(x)]) \tag{4.25}$$

Finally, if all side premises have a maximal literal of the form $R_i([f(x_i)], x_i)$, the resolvent has the form (4.26):

$$\mathbf{P}(x) \vee \mathbf{S}([f(x)], x) \tag{4.26}$$

This closure can further be decomposed into (4.27) – (4.29). Since $S_i([f(x)], x)$ and $\mathsf{Inv}(S_i)(x, [f(x)])$ are logically equivalent due to the translation operator π, the predicate $Q_{\mathsf{Inv}(S_i),f}$ can be used as the definition predicate for $S_i([f(x)], x)$.

$$\mathbf{P}(x) \vee Q_{\mathsf{Inv}(S_1),f}(x) \vee \ldots \vee Q_{\mathsf{Inv}(S_m),f}(x) \tag{4.27}$$
$$\neg Q_{\mathsf{Inv}(S_1),f}(x) \vee S_1([f(x)], x) \tag{4.28}$$
$$\vdots$$
$$\neg Q_{\mathsf{Inv}(S_m),f}(x) \vee S_m([f(x)], x) \tag{4.29}$$

We thus ensure that the conclusions of all inferences are $\mathcal{ALCHIQ}$-closures, so Lemma 4.3.5 applies for $\mathcal{ALCHIQ}b$ as well. Furthermore, the number of literals in a closure of type (4.21) is linear in the size of the role expression, so the claim of this theorem holds in the same way as for Theorem 4.4.8. $\square$

We briefly comment why role safety is important for decidability. Namely, due to safety, closures of type (4.21) always contain a negative literal which is 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 4.2, thus complicating the decision procedure. Since $\mathbf{P_1}(x)$ and $\mathbf{P_2}(y)$ do not have common variables, a possible solution is to don't-know non-deterministically assume that either disjunct is true, and thus reduce the problematic closure to an $\mathcal{ALCHIQ}$ closure. Notice that this increases the complexity of the algorithm from ExpTime to NExpTime, thus providing for the required increase; namely, in [59] it was shown that reasoning in a description logic with non-safe role expressions is NExpTime-complete. Unfortunately, the above transformation is not applicable to all problematic closures.

4.5 Related Work

Decision procedures for various logics were at the focus of the automated theorem proving research 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 [77] for first-order logic consisting of a single inference rule. Due to its simplicity, resolution soon gained popularity. Namely, a resolution-based theorem prover is not required to choose the rule to apply next, so it can easily be implemented. Soon after the initial work of Robinson, various refinements of resolution were developed, such as hyperresolution [78], ordered resolution [75] or lock resolution [18], 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 loosing completeness. A good overview of resolution and related refinements is given in the classical text-book by Chang and Lee [20].

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 [50] 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 which is incomplete for first-order logic and is therefore difficult to extend.

In [49] Joyner established the basic principles of 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 matrix 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 [85], the $\mathcal{PVD}$ class [53], and the $\mathcal{PVD}^g_=$ class [72], to name just a few. An overview of these results is given in a monograph by Fermüller, Leitsch, Tammet and Zamov [29].

Decidability of description logics in the resolution framework has been studied extensively in [67, 48, 45]. 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 have been shown 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 counting quantifiers.

In [32] 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, which produced undesirable equality literals. Therefore, we adopted the approach for eliminating transitivity from Section 4.2.

The guarded fragment was introduced in [3] to explain and generalize the good properties of modal and description logic, such as decidability. A resolution decision procedure based on a non-liftable ordering was given in [21], and was later modified to handle the (loosely) guarded fragment with equality [31] by basing the algorithm on superposition [9]. Since the basic description logic $\mathcal{ALC}$ is actually a syntactic variant of the multi-modal logic K_m [79], it can be embedded into the guarded fragment and decided by [31]. Using the approach from [81], certain extensions of $\mathcal{ALC}$, such as role transitivity, can be encoded into $\mathcal{ALC}$ knowledge bases, so the algorithm from [31] can decide these extensions as well. However, the (loosely) guarded fragment is not capable of expressing $\mathcal{SHIQ}$ because of the counting quantifiers: equality is available in the logic, but each two pairs of free variables of a guarded formula must occur in a guard atom. In fact, in [40] 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 it.

$\mathcal{SHIQ}$ 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 [34] and a decision procedure based on a combination of resolution and integer programming was given in [74]. However, deciding satisfiability of $\mathcal{C}^2$ is NExpTime-complete, and $\mathcal{SHIQ}$ is an ExpTime-complete [86] logic. Thus, the decision procedure from [74] introduces an unnecessary overhead for $\mathcal{SHIQ}$. Furthermore, we do not see how to derive the desired reduction to disjunctive datalog based on this procedure.

The decomposition rule from Subsection 4.4.1 is closely related to the structural transformation [68]. However, structural transformation is usually applied as a preprocessing step and not in the theorem proving process. In [22] and [76] *splitting by propositional symbols* was considered, which allows splitting variable-disjoint subsets of a clause and connecting them by a propositional symbol. Finally, a so-called *separation* rule, similar to decomposition, was used to decide fluted logic in [80]. It was shown that resolution remains complete if the separation rule is applied a finite number of times during saturation. Our approach differs in that we demonstrate compatibility of the decomposition rule with the standard redundancy notion. Furthermore, contrary to all approaches cited above, our rule allows decomposing a complex term into simpler terms. Finally, extending basic superposition with decomposition is not trivial, due to the non-standard approach to lifting of basic superposition.

5 Reasoning with Concrete Domains

As argued in [5], working with concrete data is essential for practical knowledge representation systems. However, existing algorithms for reasoning with concrete domains from [5, 43, 38, 57] mainly operate in tableaux and automata frameworks, which essentially work with formulae in disjunctive normal form. On the contrary, the algorithms we present in Chapter 4 are based on resolution, and they essentially work with formulae in conjunctive normal form. Hence, extending the results from Chapter 4 to combine resolution with concrete domain reasoning is not trivial.

To enable reasoning with $\mathcal{SHIQ}(\mathbf{D})$, we extend our algorithms in two stages. Firstly, in Section 5.1 we present a general approach for combining concrete domain reasoning with certain clausal calculi. The approach consists of two steps. A so-called c-*factoring* preprocessing transformation is applied first to bring a clause set into a suitable form. Next, we introduce a so-called *concrete domain resolution* inference rule which combines reasoning with concrete domains in a resolution framework. We show that this rule provides for sound and complete reasoning with concrete domains when combined with a calculus whose completeness proof is based on the model generation method. Since 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], concrete domain resolution can readily be used with any of them.

Secondly, in Section 5.2 we apply the above outlined techniques to extend the decision procedure from Chapter 4 to handle $\mathcal{SHIQ}(\mathbf{D})$. 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 with a concrete domain does not increase the reasoning complexity, i.e. it remains in ExpTime.

Results from this chapter have been published in [46]. Due to space limitations, there we did not consider the distinction between $\mathbf{D}$- and $\mathbf{d}$-satisfiability, as discussed in Subsection 5.1.2, since for $\mathcal{SHIQ}(\mathbf{D})$ without ABox assertions of the form $\neg T(a, b^{\mathsf{c}})$ these two notions coincide.

5.1 Resolution with a Concrete Domain

5.1.1 Preliminaries

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

Lemma 5.1.1. *A formula φ is $\mathbf{D}$-satisfiable if and only if $\mathsf{sk}(\varphi)$ is $\mathbf{D}$-satisfiable.*

Proof. The proof is identical to the case for ordinary satisfiability from, e.g. [30]: given a $\mathbf{D}$-model I of φ, one can build a $\mathbf{D}$-model I' of $\mathsf{sk}(\varphi)$ by extending I with the appropriate interpretations of new function symbols. Conversely, each new function symbol ensures existence of existentially implied individuals. □

Hence, to check $\mathbf{D}$-satisfiability of a formula φ, by Lemma 5.1.1 we first compute a $\mathbf{D}$-equisatisfiable set of clauses $N = \mathsf{Cls}(\varphi)$. Furthermore, since the concrete domain

$\mathbf{D}$ should be admissible, we replace each literal $\neg d(\mathbf{t})$ with $\overline{d(\mathbf{t})}$. Thus, in the rest we assume that N contains only positive literals with concrete domain predicates.

Checking formula satisfiability in any model is problematic, since one should consider models over 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 $\mathbf{D}$-models, as follows:

Definition 5.1.2. *A* Herbrand $\mathbf{D}$-interpretation *over* Σ *is a pair* (I, δ), *where* I *is a "classical" Herbrand interpretation over* Σ *and* $\delta : HU_{\mathsf{c}} \rightarrow \triangle_{\mathbf{D}}$ *is a function assigning a concrete individual to each concrete term of* HU_{c}. *With* $\delta(I)$ *we denote a* $\mathbf{D}$*-interpretation obtained form* I *by replacing each concrete term* t^{c} *with* $\delta(t^{\mathsf{c}})$. *A Herbrand* $\mathbf{D}$*-interpretation* (I, δ) *is a* Herbrand $\mathbf{D}$-model *of a set of clauses* N *if and only if* (i) $s_i \approx t_i \in I$, $1 \leq i \leq n$ *implies* $\delta(f^{\mathsf{c}}(s_1, \ldots s_n)) = \delta(f^{\mathsf{c}}(t_1, \ldots t_n))$, *for each function symbol* f^{c}, *and* (ii) $\delta(I)$ *is a* $\mathbf{D}$*-model of* N.

We now show that, analogously to the case without concrete domains, we can consider $\mathbf{D}$-satisfiability only in Herbrand $\mathbf{D}$-models.

Lemma 5.1.3. *A set of clauses* N *is* $\mathbf{D}$*-satisfiable if and only if a Herbrand* $\mathbf{D}$*-model of* N *exists.*

Proof. The $(\Leftarrow)$ direction follows immediately from Definition 5.1.2 since $\delta(I)$ is a $\mathbf{D}$-model of N. For the $(\Rightarrow)$ direction, let N be $\mathbf{D}$-satisfiable in a 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 done for first-order logic without concrete domains in [30], it is easy to see that I' is a "classical" Herbrand model of N, and that $\delta(I')$ is a $\mathbf{D}$-model of N. □

5.1.2 d-satistiability

The task of finding a Herbrand $\mathbf{D}$-model (I, δ) of a set of clauses N essentially consists of two subtasks. The first one is to find the Herbrand model I; this can be done using resolution in the standard way. To second one is to find the appropriate δ. To solve this problem, we first consider the simpler task of finding such δ which only satisfies all concrete predicates.

Definition 5.1.4. *A set of clauses* N *is* $\mathbf{d}$*-satisfiable if and only if there is a "classic" Herbrand model* I *of* N *and a valuation* $\delta : HU_{\mathsf{c}} \rightarrow \triangle_{\mathbf{D}}$ *such that* (i) $s_i \approx t_i \in I$, $1 \leq i \leq n$ *implies* $\delta(f^{\mathsf{c}}(s_1, \ldots s_n)) = \delta(f^{\mathsf{c}}(t_1, \ldots t_n))$, *for each function symbol* f^{c}, *and* (ii) $\delta(\mathbf{t}) \in d^{\mathbf{D}}$ *for each* $d(\mathbf{t}) \in I$.

Notice that, $\mathbf{d}$-satisfiability is a strictly weaker notion that $\mathbf{D}$-satisfiability: if N is $\mathbf{D}$-satisfiable, it is obviously $\mathbf{d}$-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 valuation $\delta(b^{\mathsf{c}}) = \delta(c^{\mathsf{c}}) = 1$, we have $(\delta(b^{\mathsf{c}}), \delta(c^{\mathsf{c}})) \in d^{\mathbf{D}}$, so N in $\mathbf{d}$-satisfiable. However, N is not $\mathbf{D}$-satisfiable: $\delta(I) = \{R(a, 1), d(1, 1)\}$, but then $\delta(R(a, c^{\mathsf{c}})) \in \delta(I)$ as well, so the clause $\delta(\neg R(a, c^{\mathsf{c}}))$ is not true in $\delta(I)$. Intuitively, $\mathbf{d}$-satisfiability ensures consistency of literals containing concrete domain predicates, but does not ensure consistency of literals with ordinary predicates containing both concrete and abstract terms. To give the exact relationship between **d**- and **D**-satisfiability, we introduce the notion of c-factors:

Definition 5.1.5. *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^{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 either: (i) $A|_{p_i}$ is not a variable, or (ii) $A|_{p_i}$ is a variable occurring in some negative literal in $C \setminus \{\neg A\}$, or (iii) $A|_{p_i}$ is a variable occurring in $\neg A$ at some position other than p_i.

For a clause C with n literals, let $C = C_0, \ldots, C_n$ be the sequence of clauses 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-i}. Then C_n is a c-factor *of C.*

The 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.*

Notice that a clause can have several c-factors, depending on the order in which the literals are c-factored. E.g., 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 computing a c-factor of a set of clauses N, any c-factor of C can be used to replace a clause $C \in N$.

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

Lemma 5.1.6. *Let N be a set of clauses, and N' its c-factor. Then the following claims hold:*

- *N is **D**-satisfiable if and only if N' is **D**-satisfiable.*
- *If N' is **d**-satisfiable, it is **D**-satisfiable.*

Proof. For the first claim, observe that A is equivalent with

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

Hence, the c-factor of $\neg A$ is obtained from the above formula using the application of de Morgan laws, so $\neg A$ and its c-factor are equivalent. Therefore, N and N' are equivalent, so they are **D**-equisatisfiable.

For the second claim, let I be a "classic" Herbrand model of N' and δ an assignment as in Definition 5.1.4. If $\delta(I)$ is not a **D**-model of N', then there is a clause $C \in N'$ with a ground instance $C\tau$ which 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 $\delta(A\tau) \in \delta(I)$, 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 $\delta(A_i\tau) \in \delta(I)$. 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 c-factored w.r.t. C, for all j, $A_i|_{p_{ij}}$ is a variable x^{c}_{ij} occurring in $C \setminus \{A_i\}$ only in positive literals. Let σ be a ground substitution which is identical to τ but for the mappings $x^{\mathsf{c}}_{ij} \mapsto A'_i|_{p_{ij}}$. 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 variables from $\neg A_i$ for each 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 $\delta(L'\sigma) \in \delta(I)$ and $\delta(L'\sigma) = \delta(L'\tau)$, we have $\delta(L'\tau) \in \delta(I)$, which contradicts the assumption that $\delta(C\tau)$ is false in $\delta(I)$. □

Intuitively, c-factoring allows us to "move" the concrete terms from literals without concrete predicates into the concrete domain by using concrete inequalities. In Subsection 5.1.3 we build a procedure for checking **d**-satisfiability of a set of ground clauses, which we lift to non-ground clauses in Subsection 5.1.4 and Subsection 5.1.5. Combined with c-factoring, this yields a refutation procedure for **D**-satisfiability.

5.1.3 Concrete Domain Resolution with Ground Clauses

We now develop the *ground concrete domain resolution* calculus, $\mathcal{G}^{\mathbf{D}}$ for short, for checking **d**-satisfiability of a set of clauses, where **D** is an admissible concrete domain. In order not to make the presentation too technical, we add the *concrete domain resolution* rule to the ordered resolution calculus [13] 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 a reduction ordering total on ground literals such that $\neg A \succ A$, for any atom A. 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 $C \setminus L$. We extend the literal ordering $\succ$ to clauses by identifying each clause with a multiset of literals, and compare clauses by the multiset extension of the literal ordering; we denote the clause ordering ambiguously with $\succ$. Since the literal ordering is total and well-founded on ground literals, the clause ordering is total and well-founded on ground clauses.

Definition 5.1.7. *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 non-empty 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 5.1.8. *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. □

Note that in its contrapositive form, Lemma 5.1.8 states that if S is not connected, then it is not a minimal $\mathbf{D}$-constraint. We now present the inference rules of $\mathcal{G}^{\mathbf{D}}$.

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 $\mathbf{D}$-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*. Notice 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 the fact that the concrete domain resolution rule does not have a main premise.

Definition 5.1.9. *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 ξ 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 ξ has a main premise C, then additionally $C \succ D_i$ is required.*

We now prove the soundness and completeness of $\mathcal{G}^{\mathbf{D}}$ under the standard notion of redundancy.

Lemma 5.1.10 (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, soundness 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, I is a $\mathbf{d}$-model of N only if there exists 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 $C_i \subseteq C$, C is true in I as well. □

Lemma 5.1.11 (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 some 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 $\neg A \succ A$, 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 will 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 $I_C \cup \varepsilon_C \subseteq I$, C is true in I as well. Otherwise, C can be true in $I_C \cup \varepsilon_C$ because some negative literal $\neg A$ is true in $I_C \cup \varepsilon_C$. Since $\neg A \succ A$, 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, it is false in $I_C \cup \varepsilon_C$.

Property (***): if an inference ξ with a main premise C, side premises C_i and a conclusion D is redundant in N, then there are clauses $D_i \in N$ which 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 ξ 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 non-redundant clauses for which (***) obviously holds. This property holds analogously if ξ does not have a main premise.

We now show that, if all inferences by $\mathcal{G}^{\mathbf{D}}$ from premises in N are redundant in N, then I is a **d**-model of N. The proof is by contradiction: let us assume that I is not a **d**-model of N. There may be two causes for that:

- There is a clause $C \in N$ which is false in I; such a clause is called a *counterexample* for I. Since $\succ$ is well-founded and total, we may assume w.l.o.g. that C is the smallest counterexample. C is obviously not productive, since all productive clauses are true in I. C can be non-productive 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 (***), a set of clauses $D_i \in N$ exists, 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, it must hold that $A \in I$, which is produced by some 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. By the fact that N is saturated and by (***), a set of clauses $D_i \in N$ exists, 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 minimal set of concrete domain literals $S = \{d_i(\mathbf{t_i})\}$ such that $\widehat{S}$ is $\mathbf{D}$-unsatisfiable. The literals from S must have been produced by clauses $E_i = C_i \vee d_i(\mathbf{t_i}) \in N$, where E_i is false in I_{E_i}. For any 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 (***), non-redundant 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, I is a $\mathbf{d}$-model of N, so N is $\mathbf{d}$-satisfiable. □

A *derivation* by $\mathcal{G}^{\mathbf{D}}$ from a set of clauses N_0 is a sequence of clause sets $N_0, N_1, \ldots$ where $N_i = N_{i-1} \cup \{C\}$ and C is a consequence of some inference rule of $\mathcal{G}^{\mathbf{D}}$ from premises in N_{i-1}, or $N_i = N_{i-1} \setminus \{C\}$ where C is redundant in N_{i-1}. 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 non-redundant premises in N_∞ 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_∞ is redundant in N_∞. From this and lemmata 5.1.10 and 5.1.11, we get the following result:

Theorem 5.1.12. *The set of ground clauses N is $\mathbf{d}$-unsatisfiable if and only if the limit N_∞ of a fair derivation by $\mathcal{G}^{\mathbf{D}}$ contains the empty clause.*

5.1.4 Most General Partitioning Unifiers

Lifting the concrete domain resolution rule to general clauses is not trivial, since 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 Lemma 5.1.8 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 should be checked. This is the consequence of the fact that 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 e.g. 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 $\mathbf{D}$-constraint $d(x, x)$ by unifying x and y. These examples show that, to check all potential $\mathbf{D}$-constraints at the ground level, one has to consider all possible substitutions which produce a minimal $\mathbf{D}$-constraint at the non-ground level. We formalize this idea by the following definition.

Definition 5.1.13. *Let $S = \{d_i(\mathbf{t_i})\}$ be a multiset of positive concrete domain literals. A substitution σ is a* partitioning unifier *of S if the set $S\sigma$ is connected. Furthermore, σ is a* most general partitioning unifier *if, for any partitioning unifier θ such that $\widehat{S\sigma} \equiv \widehat{S\theta}$, a substitution η exists such that $\theta = \sigma\eta$. With $\mathsf{MGPU}(S)$ we denote the set of all most general partitioning unifiers of S.*

Note that in Definition 5.1.13 we assume S is a multiset, so it can contain repeated literals. 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 several most general partitioning unifiers of S may exist. However, for a given $\widehat{S\sigma}$, all most general partitioning unifiers are identical up to variable renaming, as demonstrated next.

Let $S = \{d_i(\mathbf{t_i})\}$ be a multiset 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} denote the 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 $t_j = f(s_j^1, \ldots, s_j^m)$, where s_j^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 demonstrated next.

Lemma 5.1.14. *Let θ be a partitioning unifier of a multiset 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 is unique up to variable renaming.*

Proof. To obtain the variable x_i in C, θ must be such that $s_1^i\theta = \ldots = s_n^i\theta = T_{x_i}\theta$. 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, θ is obviously a unifier of S_C, so $\sigma = \mathsf{MGU}(S_C)$ exists and that it is unique up to variable remaining [7]. Furthermore, it is obvious that $s_1^i\sigma = \ldots = s_n^i\sigma = T_{x_i}\sigma$ and $T_{x_i}\sigma \neq T_{x_j}\sigma$ for $i \neq j$. Namely, since σ is the most general unifier of S_C, then a substitution η exists, such that $\theta = \sigma\eta$. Hence, it is impossible that $T_{x_i}\sigma = T_{x_j}\sigma$ and $T_{x_i}\sigma\eta \neq T_{x_j}\sigma\eta$. Hence, $\widehat{S\sigma} \equiv C$, so the claim of the lemma follows. □

Lemma 5.1.15. *For a multiset of concrete domain literals $S = \{d_i(\mathbf{t_i})\}$, $\mathsf{MGPU}(S)$ consists exactly of all $\mathsf{MGU}(S_C)$, for all conjunctions C over literals of S.*

Proof. For $\sigma \in \mathsf{MGPU}(S)$, $C = \widehat{S\sigma}$ is obviously a conjunction over literals of S satisfying conditions of Lemma 5.1.14, so σ is equivalent to $\mathsf{MGU}(S_C)$ up to variable renaming. Conversely, let C be a conjunction over literals of S. Now $\sigma = \mathsf{MGU}(S_C)$ is obviously a partitioning unifier of S. Let $C' = \widehat{S\sigma}$. Observe that $C \equiv C'$ does not necessarily hold: it is possible that $T_{x_i}\sigma = T_{x_j}\sigma, i \neq j$. However, C' is a conjunction over concrete domain literals satisfying conditions of Lemma 5.1.14, so $\mathsf{MGU}(S_{C'})$ exists, and it is a most general partitioning unifier of S. □

Hence, Lemma 5.1.15 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 one must examine all such conjunctions C. 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 5.2, we present such an algorithm applicable to $\mathcal{SHIQ}(\mathbf{D})$.

5.1.5 Concrete Domain Resolution with General Clauses

As usual in a resolution setting, we assume that all clauses involved in an inference rule do not share common variables. The inference rules of the *concrete domain resolution* calculus, $\mathcal{R}^{\mathbf{D}}$ for short, are presented below.

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 multiset $S = \{d_i(\mathbf{t_i})\}$, $\sigma \in \mathsf{MGPU}(S)$, (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 the constraint (i) of the concrete domain resolution rule. Consider a set of clauses $N = \{d(x, y), a \not\approx_{\mathbf{D}} c\}$. Assuming that $d^{\mathbf{D}} = \{(1, 2), (2, 1)\}$, for $S = \{d(a, b), d(b, c), a \not\approx_{\mathbf{D}} c\}$ of ground instances of $d(x, y)$, the conjunction $\widehat{S}$ is $\mathbf{D}$-unsatisfiable. This is detected by the concrete domain resolution rule only if several "copies" of the clause $d(x, y)$ are considered simultaneously. Without any assumptions on the nature of the predicate d, 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 5.2 we show that the number of "copies" to 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 non-ground derivation.

Lemma 5.1.16 (Lifting)**.** *Let N be a set of clauses with the set of ground instances N^G. For each ground inference ξ^G by $\mathcal{G}^{\mathbf{D}}$ applicable to premises $C_i^G \in N^G$, there is an inference ξ by $\mathcal{R}^{\mathbf{D}}$ applicable to premises $C_i \in N$, where ξ^G is an instance of ξ.*

Proof. Let C_i^G be ground premises from N^G participating in ξ^G, resulting in a ground clause D^G. The ground inference ξ^G of $\mathcal{G}^{\mathbf{D}}$ can be simulated by a corresponding non-ground inference ξ, where for each ground premise C_i^G we take the corresponding non-ground premise C_i that C_i^G is an instance of. Since all C_i are variable-disjoint, there is a ground substitution τ such that $C_i^G = C_i\tau$. Let us denote with D the result of ξ on C_i. We now show that ξ is an inference of $\mathcal{R}^{\mathbf{D}}$.

Let ξ^G be an inference by positive factoring on ground literals A_i^G of C^G. Substitution τ is obviously a unifier of corresponding non-ground literals A_i of C. Since any unifier is an instance of the most general unifier σ of A_i, a substitution η 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, corresponding A_i is strictly maximal with respect to $C\sigma$, so $D^G = D\eta$. Hence, ξ is an inference of $\mathcal{R}^{\mathbf{D}}$. Similar reasoning applies in the case of ordered resolution.

Let ξ^G be an inference by concrete domain resolution, and let $S = \{d_i(\mathbf{t_i})\}$ be the set of corresponding non-ground literals. Since $S\tau$ is a minimal $\mathbf{D}$-constraint, by Lemma 5.1.8 $S\tau$ is connected, so τ is obviously a partitioning unifier of S. Then, by Lemma 5.1.14 there exists a most general partitioning unifier σ such that $\tau = \sigma\eta$ for some η, 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, corresponding literals from $S\sigma$ are strictly maximal with respect to $C_i\sigma$, so $D^G = D\eta$. Hence, ξ is an inference of $\mathcal{R}^{\mathbf{D}}$. □

The notion of redundancy is lifted to the non-ground case as usual [13]: a clause C (an inference ξ) is redundant in a set of clauses N if all ground instances of C (ξ) are redundant in N^G. This is enough for soundness and completeness of $\mathcal{R}^{\mathbf{D}}$.

Theorem 5.1.17. *The set of clauses N is* $\mathbf{d}$*-unsatisfiable if and only if the limit N_∞ of a fair derivation by $\mathcal{R}^{\mathbf{D}}$ contains the empty clause.*

Proof. The set N is $\mathbf{d}$-unsatisfiable if and only if the set of its ground instances N^G is $\mathbf{d}$-unsatisfiable. By Theorem 5.1.12, 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_∞^G contains the empty clause. By Lemma 5.1.16 and the definition of the redundancy for non-ground clauses, for each ground derivation, a non-ground derivation $N = N_0, N_1, \ldots, N_\infty$ exists, where each N_i^G is a subset of the set of ground instances of N_i. Hence, N_∞^G contains the empty clause if and only if N_∞ contains the empty clause, so the claim follows. □

To obtain a refutation procedure for checking $\mathbf{D}$-satisfiability of a set of clauses N, we first compute its c-factor N'; by Lemma 5.1.6, N is $\mathbf{D}$-satisfiable if and only if N' is $\mathbf{d}$-satisfiable. The latter can be checked using $\mathcal{R}^{\mathbf{D}}$.

5.1.6 Deleting D-tautologies

In ordinary resolution, clauses which are true in any model can be removed without jeopardizing completeness of the calculus. Removing such clauses is crucial for practical applicability of resolution, since it drastically reduces the search space. To achieve the same effect for $\mathcal{R}^{\mathbf{D}}$, in this subsection we define the notion of $\mathbf{D}$-tautologies as clauses which are true in any $\mathbf{D}$-model due to properties of the concrete domain. We show that $\mathbf{D}$-tautologies can be deleted eagerly in the saturation process while preserving completeness.

Definition 5.1.18. *A clause is a* $\mathbf{D}$*-tautology if it contains a* $\mathbf{D}$*-tautology literal $d(\mathbf{t})$; the latter is the case if and only if $\widehat{d(\mathbf{t})} = d(\mathbf{x})$ is true for any assignment δ of variables to elements of $\triangle_{\mathbf{D}}$.*

For example, the clauses $C \vee f(x) \approx_{\mathbf{D}} f(x)$ and $C \vee \top_{\mathbf{D}}(x)$ are $\mathbf{D}$-tautologies. Similarly, if $\triangle_{\mathbf{D}}$ is the set of non-negative integers, then the clause $C \vee \geq_0 (x)$ is also a $\mathbf{D}$-tautology. We now show that such clauses are redundant in the saturation process and can be deleted.

Lemma 5.1.19. *$\mathbf{D}$-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 $\mathbf{D}$-constraint. Obviously, S cannot contain a $\mathbf{D}$-tautology literal $d(\mathbf{t})$, since $S \setminus \{d(\mathbf{t})\}$ would be an even smaller $\mathbf{D}$-constraint. Hence, $\mathbf{D}$-tautologies do not participate in 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 5.1.11 to all clauses from N which are not $\mathbf{D}$-tautologies, and let δ be an assignment of elements from $\triangle_{\mathbf{D}}$ to elements of HU_{c} (such an assignment exists by Lemma 5.1.11 since 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 $\mathbf{D}$-tautology literal $d(\mathbf{t})$ of a ground clause $C \in N$. Since all concrete domain literals occur positively in clauses in N, adding 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', δ) is a Herbrand $\mathbf{d}$-model of N. Effectively, a $\mathbf{D}$-tautology C generates in I' only $\mathbf{D}$-tautology concrete literals which cannot participate in a concrete domain resolution rule; thus C can be deleted from N.

For the non-ground calculus $\mathcal{R}^{\mathbf{D}}$, simply observe that if a non-ground literal $d(\mathbf{t})$ is a $\mathbf{D}$-tautology, then for all ground substitutions τ, the literal $d(\mathbf{t})\tau$ is a $\mathbf{D}$-tautology as well, so the lifting argument from Lemma 5.1.16 holds without change. $\square$

We note that a clause containing a complementary pairs of concrete literals is *not* a $\mathbf{D}$-tautology and should not be deleted. Consider the following $\mathbf{d}$-unsatisfiable set of clauses (recall that $\approx_{\mathbf{D}}$ is just a special concrete domain predicate):

$$a \approx_{\mathbf{D}} b \vee a \approx_{\mathbf{D}} c \tag{5.1}$$
$$b \approx_{\mathbf{D}} c \tag{5.2}$$
$$a \not\approx_{\mathbf{D}} b \vee a \not\approx_{\mathbf{D}} c \tag{5.3}$$

Let $a >_P b >_P c >_P \not\approx_{\mathbf{D}} >_P \approx_{\mathbf{D}}$; the maximal literal in a clause is denoted like this. We demonstrate $\mathbf{d}$-unsatisfiability in the following way.

$$a \approx_{\mathbf{D}} c \vee a \not\approx_{\mathbf{D}} c \tag{5.4}$$
$$a \approx_{\mathbf{D}} c \tag{5.5}$$
$$a \not\approx_{\mathbf{D}} c \tag{5.6}$$
$$\square \tag{5.7}$$

Here, (5.4) is derived from (5.1) and (5.3) since $\{a \approx_{\mathbf{D}} b, a \not\approx_{\mathbf{D}} b\}$ is a $\mathbf{D}$-constraint; (5.5) is derived from (5.1), (5.2) and (5.4) since $\{a \approx_{\mathbf{D}} b, b \approx_{\mathbf{D}} c, a \not\approx_{\mathbf{D}} c\}$ is a $\mathbf{D}$-constraint; (5.6) is derived from (5.2), (5.3) and (5.5) since $\{b \approx_{\mathbf{D}} c, a \not\approx_{\mathbf{D}} b, a \approx_{\mathbf{D}} c\}$ is a $\mathbf{D}$-constraint; and (5.7) is derived from (5.5) and (5.6) since $\{a \approx_{\mathbf{D}} c, a \not\approx_{\mathbf{D}} c\}$ is a $\mathbf{D}$-constraint.

Notice that (5.4) contains a complementary pair of concrete literals, but it should not be deleted; 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.

5.1.7 Combining Concrete Domains with Other Resolution Calculi

Observe that the proof of Lemma 5.1.6 is model-theoretic and does not depend on any calculus. Hence, c-factoring can always be applied as a preprocessing step.

In order not to make the presentation too technical, we extended only the ordered resolution calculus with the concrete domain resolution rule. However, from the proof of Lemma 5.1.11, one may see that the concrete domain resolution rule is largely independent from the actual calculus, and may 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 which can have at least one productive ground instance. For the ordered resolution with selection, this means that premises are not allowed to contain selected literals (since such clauses do not have productive ground instances). The usual arguments for the calculus at hand show that, if N_∞ does not contain the empty clause, one may generate an interpretation I using the model generation method, such that all clauses from N_∞ are true in I. Furthermore, the argument from the second part of Lemma 5.1.11 shows independently that, if all inferences by the concrete domain resolution rule are redundant in N_∞, then I is a $\mathbf{d}$-model of N.

In the rest, with $\mathcal{BS}^{\mathbf{D}}$ we denote the extension of the basic superposition calculus with a concrete domain resolution rule. Notice that $\approx_{\mathbf{D}}$ is a predicate, so due to encoding of predicate symbols by function symbols cf. Section 2.4, the literal $s \approx_{\mathbf{D}} t$ is actually a shortcut for $(s \approx_{\mathbf{D}} t) \approx \top$.

Theorem 5.1.20. *Let N_∞ 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_∞ does not contain the empty closure.*

Proof. The soundness is demonstrated in the same way as in Lemma 5.1.10. For completeness, let us assume that N_∞ does not contain the empty closure. By [14, 65], 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 5.1.11, $\mathsf{irred}_{R_N}(N)$ is $\mathbf{d}$-satisfiable, i.e. there is an function δ assigning concrete individuals to concrete terms from $\mathsf{irred}_{R_N}(N)$ such that, for each $d(\mathbf{t}) \in {R_N}^*$, $\delta(\mathbf{t}) \in d^{\mathbf{D}}$.

The concrete domain resolution rule does not ensure that the assignment δ satisfies condition (i) of Definition 5.1.4, i.e. it may be that there are terms $s_i \approx t_i \in {R_N}^*$, but for some function symbol f^{c}, we have $\delta(f^{\mathsf{c}}(s_1, \dots, s_n)) \neq \delta(f^{\mathsf{c}}(t_1, \dots, t_n))$. Consider an assignment δ' obtained from δ by setting $\delta'(u) = \delta(\mathsf{nf}_{R_N}(u))$. Obviously, δ' now fulfills the condition (i) of Definition 5.1.4. Furthermore, ${R_N}^*$ is an equality interpretation, so if $d(\mathbf{t}) \in {R_N}^*$, then $d(\mathbf{t}') \in {R_N}^*$ as well, where $\mathbf{t}' = \mathsf{nf}_{R_N}(\mathbf{t})$. Hence, since δ fulfills condition (ii) of Definition 5.1.4, δ' fulfills it as well, so $({R_N}^*, \delta')$ is a $\mathbf{d}$-model of $\mathsf{irred}_{R_N}(N)$.

Now the assignment δ' can be extended to all ground concrete terms by ensuring that $s_i \approx t_i \in {R_N}^*$, $1 \leq i \leq n$, implies $\delta'(f^{\mathsf{c}}(s_1, \dots, s_n)) = \delta'(f^{\mathsf{c}}(t_1, \dots, t_n))$, for any function symbol f^{c}. Obviously, $({R_N}^*, \delta')$ is a $\mathbf{d}$-model of N. □

The above proof ensures by a model-theoretic argument that the properties of the model related to equality (condition (i) of Definition 5.1.2) are satisfies. To develop the intuition behind this result, we present an alternative, proof-theoretic argument. Condition (i) of Definition 5.1.2 is obviously fulfilled if, for each function symbol f^{c} of arity n, we add the following closure to the closure set:

$$x_1 \not\approx y_1 \vee \ldots \vee x_n \not\approx y_n \vee f^{\mathsf{c}}(x_1, \ldots, x_n) \approx_{\mathbf{D}} f^{\mathsf{c}}(y_1, \ldots, y_n)$$

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:

$$f^{\mathsf{c}}(x_1, \ldots, x_n) \approx_{\mathbf{D}} f^{\mathsf{c}}(x_1, \ldots, x_n)$$

Such closures are obviously $\mathbf{D}$-tautologies, so by Lemma 5.1.19 they can be removed from the closure set. Effectively, Condition (i) of Definition 5.1.2 is always trivially satisfied. However, this condition still is needed in Definition 5.1.2, since it ensures that abstract equality is interpreted for concrete terms in the usual way.

5.2 Deciding $\mathcal{SHIQ}(\mathbf{D})$

In this section we combine the concrete domain resolution rule from Section 5.1 with the algorithm from Chapter 4 to obtain a decision procedure for checking satisfiability of $\mathcal{SHIQ}(\mathbf{D})$ knowledge bases. We also show that this extension does not increase the complexity of reasoning, assuming a bound on the arity of concrete predicates and a bound on the complexity of the oracle for checking $\mathbf{D}$-satisfiability of concrete conjunctions.

It is easy to see that the operator Ω from Section 4.2 can be used to eliminate transitivity axioms from a $\mathcal{SHIQ}(\mathbf{D})$ knowledge base KB by encoding in into an equisatisfiable knowledge base $\Omega(KB)$. Namely, concrete roles cannot be transitive, so Theorem 4.2.3 applies without changes. Hence, without loss of generality, in the rest of this section we focus on deciding satisfiability of $\mathcal{ALCHIQ}(\mathbf{D})$ knowledge bases.

With $\mathcal{BS}_{DL}^{\mathbf{D},+}$ we denote the $\mathcal{BS}^{+}$ calculus extended with the concrete domain resolution rule, parameterized as specified in Definition 4.3.3. To obtain a decision procedure for $\mathcal{ALCHIQ}(\mathbf{D})$, we show that the results of Lemma 4.3.5 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 lead to generation of 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.

5.2.1 Closures with Concrete Predicates

As before, with $\Xi(KB)$ we denote the set of closures obtained from KB by structural transformation, as explained in Definition 4.3.1. By definition of π from Table 3.1 and Table 3.3, it is easy to see that $\Xi(KB)$ may contain closures with structure as in Table 4.1, with all variables, function symbols and predicate arguments being of the sort a, and additionally closures with structure as in Table 5.1. Notice that closures of type 21 are obtained by c-factoring of closures of type $\neg T(a, b^{\mathsf{c}})$.

We now generalize the closures from Table 5.1 to include closure types produced in a saturation of $\Xi(KB)$ by $\mathcal{BS}_{DL}^{\mathbf{D},+}$. So called $\mathcal{ALCHIQ}(\mathbf{D})$-closures include those with structure as in tables 4.2 and 5.2. By the definition of Ξ, it is obvious that Lemma

Table 5.1: Additional Closures after Preprocessing

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	$\bigvee(\neg)C_i(x) \vee \bigvee \neg T_i(x, y_i^{\mathsf{c}}) \vee d(y_1^{\mathsf{c}}, \ldots, y_m^{\mathsf{c}})$
17	$\bigvee(\neg)C_i(x) \vee \bigvee_{i=1}^{n} \neg T(x, y_i^{\mathsf{c}}) \vee \bigvee_{i,j=1;j>i}^{n} y_i^{\mathsf{c}} \approx_{\mathbf{D}} y_j^{\mathsf{c}}$
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	$a^{\mathsf{c}} \not\approx_{\mathbf{D}} b^{\mathsf{c}}$

4.3.2 and Lemma 4.3.4 hold for $\mathcal{ALCHIQ}(\mathbf{D})$-closures as well. To make a distinction with generators of type 3, we call closures of type 10 where $f^{\mathsf{c}}(x)$ occurs unmarked c-*generators*.

5.2.2 Closure of $\mathcal{ALCHIQ}(\mathbf{D})$-closures under Inferences

We now extend Lemma 4.3.5 to handle $\mathcal{ALCHIQ}(\mathbf{D})$-closures.

Lemma 5.2.1. *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. Since the sorts of the abstract and concrete domain predicates are disjoint, and since in closures of type 11 literals with concrete predicates or concrete equalities are always maximal, an inference between closures of types 1 – 7 and 9 – 13 is not possible. Furthermore, a closure of type 8 can participate in an inference with a closure of type 1 – 7 only on abstract, and with a closure of type 9 – 13 only on concrete predicates. Hence, the proof of Lemma 4.3.5 remains valid for closures of types 1 – 8. Furthermore, decomposition can be applied analogously as in Theorem 4.4.8.

Table 5.2: Types of $\mathcal{ALCHIQ}(\mathbf{D})$-closures

8	in addition to literals from Table 4.2, a closure can additionally contain $\ldots \mathbf{T}(\langle \mathbf{a} \rangle, \langle \mathbf{b}^{\mathsf{c}} \rangle) \vee \mathbf{d}(\langle \mathbf{t}_1^{\mathsf{c}} \rangle, \ldots, \langle \mathbf{t}_n^{\mathsf{c}} \rangle) \vee \bigvee \langle t_i^{\mathsf{c}} \rangle \circ \langle t^{\mathsf{c}}{}_j \rangle \vee \bigvee \neg T(\langle a \rangle, y^{\mathsf{c}}) \vee y^{\mathsf{c}} \not\approx_{\mathbf{D}} b^{\mathsf{c}}$ where t_i^{c} and t_j^{c} are either some constant b^{c} or a functional term $f_i^{\mathsf{c}}([a])$
9	$\neg T(x, y^{\mathsf{c}}) \vee U(x, y^{\mathsf{c}})$
10	$\mathbf{P}^{f^{\mathsf{c}}}(x) \vee T(x, \langle f^{\mathsf{c}}(x) \rangle)$
11	$\mathbf{P}(x) \vee \mathbf{d}(\langle \mathbf{f}_1^{\mathsf{c}}(x) \rangle, \ldots, \langle \mathbf{f}_n^{\mathsf{c}}(x) \rangle) \vee \bigvee \langle f_i^{\mathsf{c}}(x) \rangle \circ \langle f^{\mathsf{c}}{}_j(x) \rangle$
12	$\mathbf{P}(x) \vee \bigvee \neg T(x, y_i^{\mathsf{c}}) \vee \bigvee y_i^{\mathsf{c}} \approx_{\mathbf{D}} y_j^{\mathsf{c}}$
13	$\mathbf{P}(x) \vee \bigvee \neg T_i(x, y_i^{\mathsf{c}}) \vee d(y_1^{\mathsf{c}}, \ldots, y_n^{\mathsf{c}})$
Note: $\circ \in \{\approx_{\mathbf{D}}, \not\approx_{\mathbf{D}}\}$	

Since equality among concrete terms is actually a concrete predicate, it does not participate in superposition inferences; it participates only in concrete domain resolution. Hence, there are no superposition inferences into c-generators, so there are no termination problems with such inferences as in Lemma 4.3.5. Finally, in a closure of type 8 containing a literal $\neg T(\langle a \rangle, y^c)$, this literal is selected. Hence, such a closure can only participate in superposition into the first argument, or in resolution with a closure of type 3. In both cases, the result is a closure of type 8.

The most important difference to the proof of Lemma 4.3.5 lies in the application of the concrete domain resolution rule, with n side premises of the form $C_i \vee d_i(\langle \mathbf{t_i} \rangle)$. For side premises of type 11, we denote their free variables with x_i. Let $S = \{d_i(\mathbf{t_i})\}$. For any most general partitioning unifier σ of S, $S\sigma$ must be connected. Hence, there are two possibilities:

- If all side premises are of type 11, σ is of the form $\{x_2 \mapsto x_1, \ldots, x_n \mapsto x_1\}$. The result is obviously a closure of type 11.
- 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 $d_i(\mathbf{f^c}_\mathbf{i}(x_i))$ would not contain a ground term, so $S\sigma$ would not be connected. Hence, all literals from $S\sigma$ are ground, and σ 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 non-redundant inferences of $\mathcal{BS}_{DL}^{\mathbf{D},+}$ applied to $\mathcal{ALCHIQ}(\mathbf{D})$-closures produce an $\mathcal{ALCHIQ}(\mathbf{D})$-closure. □

By inspecting the proof of Lemma 5.2.1, one may easily extend the Corollary 4.3.6 to include $\mathcal{ALCHIQ}(\mathbf{D})$ closures:

Corollary 5.2.2. *If a closure of type 8 participates in a $\mathcal{BS}_{DL}^{\mathbf{D},+}$ inference in a derivation from Lemma 5.2.1, the unifier σ 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.*

5.2.3 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. As in Lemma 4.3.8, f is linear in $|KB|$ for unary coding of numbers, since by skolemizing $\exists T_1, \ldots, T_m.d$ we introduce m function symbols. In addition to literals from $\mathcal{ALCHIQ}$-closures, $\mathcal{ALCHIQ}(\mathbf{D})$-closures may contain literals of the form $d(\langle f_1(x) \rangle, \ldots, \langle f_m(x) \rangle)$. The maximal non-ground closure will contain all such literals, of which there can be $d(2f)^m$ many (the factor 2 takes into account that each functional term can be marked or not). Hence, 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 becomes doubly-exponential, thus invalidating the results of Lemma 4.3.8.

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. In this case, m becomes a constant, and does not depend on $|KB|$. The maximal length of a closure is then polynomial in $|KB|$, and the number of closures is exponential in $|KB|$. Hence, Lemma 4.3.8 can be easily extended to include $\mathcal{ALCHIQ}(\mathbf{D})$-closures. The following lemma shows the last step in determining the complexity of the decision algorithm.

Lemma 5.2.3. *The maximal number of side premises participating in a concrete domain resolution rule in a $\mathcal{BS}_{DL}^{\mathbf{D},+}$ derivation from Lemma 5.2.1 is at most polynomial in $|KB|$, assuming a limit 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 σ 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 where 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 non-ground 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 ℓ_{ng} and ℓ_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 σ 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 ℓ_{ng}. Thus, the maximal number of non-ground side premises n to be considered is bounded by ℓ_{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, σ 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 ℓ_g. Thus, the maximal number of side premises n (by counting each “copy” of a premise separately) to be considered is bounded by ℓ_g. □

Theorem 5.2.4. *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 4.3.8 and Theorem 4.4.8. By substituting Lemma 4.3.5 for Lemma 5.2.1, the proof of the Theorem 4.3.9 can straightforwardly be extended to 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 5.2.3, $\ell = |N'|$ is polynomial in $|KB|$, and N' contains at most ℓ variables. If N' does not contain a closure of type 8, there is exactly one substitution σ which unifies all ℓ variables. If there is at least one closure of type 8, then each one of ℓ variables can be assigned to one of c constants, producing c^ℓ 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 5.2.3 demonstrates how to implement the concrete domain resolution rule in practice. There are two cases:

- For concrete domain resolution without closures 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, one should consider only closures with maximal literals unique up to variable renaming, so several "copies" of a closure need not be considered.

- For concrete domain resolution where at least one closure is of type 8, one first chooses 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 functional 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 σ of $\Delta^g{}_0 \cup \Delta^{ng}{}_0$ will contain 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 σ, 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 functional term $f(c)$ in $\Delta^g{}_1$, we have $\Delta^f{}_0 \subsetneq \Delta^f{}_1$; in this case we repeat the above process. 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.

5.3 Related Work

The need to represent and reason with concrete data was recognized in description logic systems early on. The early systems such as MESON [25] and CLASSIC [16] provided such features, mostly by means of built-in predicates.

The first rigorous treatment of reasoning with concrete data in description logics was given 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, i.e. it remains in PSPACE. Sound and complete reasoning can be performed by extending the standard tableaux 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 [1].

Contrary to modern description logic systems, the approach from [5] does not allow TBoxes. In [58] it was shown that allowing cyclic TBoxes makes reasoning with a concrete domain undecidable in general. More importantly, undecidability holds for all numeric concrete domains, which are probably the most practically relevant ones. Still, in [57] 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 [58] 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 feature chains, which may be used to force equivalence of objects at the end of two distinct feature chains. Hence, feature chains with concrete domains destroy the tree-model property, which was identified in [87] 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 [38], 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 is presented in [38], obtained by extending the tableaux algorithm with concrete domain constraint checking. In [43] the term *datatypes* was introduced for a concrete domain without feature chains, and a tableaux 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 [70]. The results of this research influenced significantly the development of the Semantic Web ontology language OWL [71], 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 by the *datatype groups* approach in [69], which additionally simplifies the integration process by taking care of the extension of the negative concrete literals.

Until now, all existing approaches consider reasoning with a concrete domain in tableaux and automata frameworks. In the resolution setting, many Prolog-like systems, such as XSB[1], provide built-in predicates to handle elementary 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 which supports reasoning with a concrete domain in the resolution setting and is complete w.r.t. semantics from [5].

Concrete domain resolution calculus can be viewed as an instance of theory resolution [84]. 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 related to concrete domains,

[1] http://xsb.sourceforge.net/

such as the necessity to consider several "copies" of a clause in a concrete domain resolution inference.

6 Reducing DL to Disjunctive Datalog

Based on the decision procedure for $\mathcal{SHIQ}(\mathbf{D})$ from Chapter 4 and Chapter 5, 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 for query answering. Furthermore, we also present an algorithm for query answering in $\mathsf{DD}(KB)$ running in worst-case exponential time, and thus show that query answering by reduction to disjunctive datalog is worst-case optimal.

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".

6.1 Overview

Since $\mathcal{SHIQ}(\mathbf{D})$ is actually a fragment of first-order logic, any $\mathcal{SHIQ}(\mathbf{D})$ knowledge base KB can be converted into a set of first-order clauses $\mathsf{Cls}(\pi(KB))$ by well-known transformations. These clauses can then be converted into rules by moving all positive literals to the rule head, and all negative literals to the rule body. However, such a naïve approach of converting KB to a rule setting is far from satisfactory. Consider the knowledge base KB containing the only axiom $C \sqsubseteq \exists R.C$; the "rules" obtained by applying the above procedure to KB are shown below:

$$R(x, f(x)) \leftarrow C(x)$$
$$C(f(x)) \leftarrow C(x)$$

These rules contain a functional term $f(x)$, obtained by skolemizing the existential quantifier. It is well-known that query answering for such rules is not decidable. Available query answering techniques for rules with function symbols, such as SLD-resolution [55], are not guaranteed to terminate on such rules. Hence, such a naïve reduction into rules destroys decidability, the most prominent property of most modern description logics.

Contrary to the simple translation into rules, our goal is to obtain a reduction which produces (disjunctive) programs without function symbols. Disjunctive datalog is itself a decidable formalism, with known terminating query answering algorithms. Hence, reduction to datalog fully preserves decidability.

Our reduction is based on the observation that, after performing all $\mathcal{BS}_{DL}^{\mathbf{D},+}$ inferences with non-ground clauses, all remaining inferences involve a clause of type 8 containing functional terms of depth at most one. Intuitively, we simulate these terms with fresh constants. This enables reducing each $\mathcal{BS}_{DL}^{\mathbf{D},+}$ refutation from $\Xi(KB)$ to a refutation in disjunctive datalog. More closely, the reduction algorithm proceeds by the following steps:

- Transitivity axioms are eliminated using the transformation from Section 4.2. Hence, in the rest we focus on $\mathcal{ALCHIQ}(\mathbf{D})$ knowledge bases only.
- The TBox and RBox clauses of $\Xi(KB)$ are saturated together with $\mathsf{gen}(KB)$ by $\mathcal{BS}_{DL}^{\mathbf{D},+}$. Hence, in this step we perform all $\mathcal{BS}_{DL}^{\mathbf{D},+}$ inference steps with non-ground clauses. As shown by Lemma 6.2.1, certain clauses can be removed from the saturated set, as they may not participate in further inferences.

- If the saturated set does not contain the empty clause, function symbols are eliminated from saturated clauses cf. Section 6.2. Lemma 6.2.4 demonstrates that this transformation does not affect satisfiability. Intuitively, this is so because (i) if ABox clauses are added to the saturated set and saturation is continued, all remaining inferences involve a clause of type 8 and produce such a clause and (ii) 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 may be removed, cf. Section 6.3. Lemma 6.3.2 demonstrates that this transformation also does not affect satisfiability.

- Transformation of KB into a disjunctive datalog program, cf. Section 6.4, is now straightforward: it suffices to transform each clause into the equivalent sequent form. This transformation does not affect entailment, which is trivially demonstrated by Theorem 6.4.2.

6.2 Eliminating Function Symbols

For an $\mathcal{ALCHIQ}(\mathbf{D})$ knowledge base KB, let $\Gamma_{\mathcal{TR}g} = \Xi(KB_{\mathcal{T}} \cup KB_{\mathcal{R}}) \cup \mathsf{gen}(KB)$. Let $\mathsf{Sat}_{\mathsf{R}}(\Gamma_{\mathcal{TR}g})$ denote the *relevant set of saturated clauses*, that is, clauses of type 1, 2, 3, 5, 7 and 9 – 13 obtained by saturating $\Gamma_{\mathcal{TR}g}$ using $\mathcal{BS}_{DL}^{\mathbf{D},+}$ with eager application of redundancy elimination rules. Finally, let $\Gamma = \mathsf{Sat}_{\mathsf{R}}(\Gamma_{\mathcal{TR}g}) \cup \Xi(KB_{\mathcal{A}})$. Intuitively, $\mathsf{Sat}(\Gamma_{\mathcal{TR}g})$ contains all non-redundant clauses derivable from TBox and RBox by $\mathcal{BS}_{DL}^{\mathbf{D},+}$. Hence, any inference in the saturation of Γ will involve 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 these clauses and consider only the $\mathsf{Sat}_{\mathsf{R}}(\Gamma_{\mathcal{TR}g})$ subset. Adding $\mathsf{gen}(KB)$ is necessary to compute all non-ground consequences of clauses possibly introduced by decomposition.

Lemma 6.2.1. *KB is $\mathbf{D}$-unsatisfiable if and only if Γ is $\mathbf{D}$-unsatisfiable.*

Proof. KB is $\mathbf{D}$-equisatisfiable with $\Xi(KB)$. Let $\Gamma' = \Xi(KB) \cup \mathsf{gen}(KB)$. 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 Γ' by adjusting the interpretation of $Q_{R,f}$ as needed. Hence, $\Xi(KB)$ is $\mathbf{D}$-equisatisfiable with Γ'. Γ' is $\mathbf{D}$-unsatisfiable if and only if the set of clauses derived by saturating Γ' with $\mathcal{BS}_{DL}^{\mathbf{D},+}$ contains the empty clause. Since the order in which the inferences are performed can be chosen don't-care non-deterministically, we perform all non-redundant inferences among clauses from $\Gamma_{\mathcal{TR}g}$ first. Let us denote the resulting set of intermediate clauses with $N_i = \mathsf{Sat}(\Gamma_{\mathcal{TR}g}) \cup \Xi(KB_{\mathcal{A}})$.

If N_i contains the empty clause, Γ contains it as well by definition (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 easily show that $N_j \setminus N_i$ contains only clauses of type 8: namely, all non-redundant inferences between clauses of type other than 8 have been performed in N_i, and, by Corollary 5.2.2, 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, according to Lemma 4.4.10, thus obtained clause of type 3 has the form $\neg Q_{R,f}(x) \vee R(x, [f(x)])$ and is in $\mathsf{gen}(KB)$, so only a clause of type 8 is added to N_j.

Furthermore, by Corollary 5.2.2, 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 Γ. Any set of clauses N_j, $j > i$, which can be obtained by saturation from Γ', can be obtained by saturation from Γ as well, modulo clauses of type 4 and 6. Hence, the saturation of Γ' by $\mathcal{BS}_{DL}^{\mathbf{D},+}$ derives the empty clause if and only if the saturation of Γ 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 4.3.7, clauses of types 3 or 5 containing a functional term cannot participate in an inference with a clause of type 8. Hence, such clauses can also be eliminated from the saturated set and 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 literal containing a functional term.

We now show how to eliminate function symbols from clauses in Γ. Intuitively, the idea is to replace each ground functional term $f(a)$ with a new constant, denoted as a_f. For each function symbol f we introduce a new predicate symbol S_f, containing, for each 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 to replace each occurrence of a term $f(x)$ with a new variable x_f and, for each such x_f, to append the literal $\neg S_f(x, x_f)$ to C. Under this transformation, the Herbrand universe of the clause set becomes finite, and can be represented by a predicate symbol HU whose extension contains all constants a and a_f. The predicate symbol HU is used to bind unsafe variables in clauses. We formalize this process by defining the operator λ as follows:

Definition 6.2.2. *Let KB be an $\mathcal{ALCHIQ}(\mathbf{D})$ knowledge base. The operator λ is defined on the set of terms and produces a term as follows:*

- $\lambda(a) = a$.
- $\lambda(f(a)) = a_f$, *where a_f is a new globally unique constant*[1].
- $\lambda(x) = x$.
- $\lambda(f(x)) = x_f$, *where x_f is a new globally unique variable.*

We extend λ 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 by $\lambda(t)$.*
2. *For each variable x_f introduced in the first step, the literal $\neg S_f(x, x_f)$ is appended to the clause.*
3. *If after steps 1 and 2 some variable x occurs in a positive literal, but not in a negative literal, the literal $\neg HU(x)$ is appended to the clause.*

[1]Globally unique means that, for some f and a, the constant a_f is always the one and the same.

For a position p in a clause C, let $\lambda(p)$ denote the corresponding position in $\lambda(C)$. For a substitution σ, let $\lambda(\sigma)$ denote the substitution obtained from σ by replacing each assignment $x \mapsto t$ with $x \mapsto \lambda(t)$. Let λ^{-} denote the inverse of λ (i.e. $\lambda^{-}(\lambda(\alpha)) = \alpha$ for any term, clause, position or a substitution α)[2].

Let $\mathsf{FF}(KB) = \mathsf{FF}_{\lambda}(KB) \cup \mathsf{FF}_{Succ}(KB) \cup \mathsf{FF}_{HU}(KB) \cup \Xi(KB_{\mathcal{A}})$ denote the function-free version of $\Xi(KB)$, where FF_{λ}, FF_{Succ} and FF_{HU} are defined as follows, with a and f ranging over all constant and function symbols in $\Xi(KB)$, and C ranging over all clauses in $\mathsf{Sat}_{\mathsf{R}}(\Gamma_{\mathcal{TR}g})$:

$$\begin{aligned}\mathsf{FF}_{\lambda}(KB) &= \bigcup \lambda(C)\\ \mathsf{FF}_{Succ}(KB) &= \bigcup S_f(a, \lambda(f(a)))\\ \mathsf{FF}_{HU}(KB) &= \bigcup HU(a) \cup \bigcup HU(\lambda(f(a)))\end{aligned}$$

Corollary 6.2.3. *Assuming $\Xi(KB_{\mathcal{A}})$ is c-factored, $\mathsf{FF}(KB)$ is c-factored as well. Furthermore, non-ground negative equality literals occur in clauses of $\mathsf{FF}(KB)$ only within disjunctions of the form $x_f \not\approx x_f \vee \neg S_f(x, x_f) \vee \neg S_g(x, x_g)$*

Proof. For the first claim, observe that each clause C of type 9, 12 and 13 is c-factored, so $\lambda(C)$ is c-factored as well. Furthermore, for a clause C of type 10 and 11, a functional term of sort c occurs only in positive literals of C. Hence, terms of sort c occur in $\lambda(C)$ only in negative literals of the form $\neg S_f(x, x_f^{\mathsf{c}})$. Moreover, each variable x_f^{c} occurs in exactly one such literal, so $\lambda(C)$ is c-factored. Hence, if $\Xi(KB_{\mathcal{A}})$ is c-factored, then $\mathsf{FF}(KB)$ is c-factored as well.

For the second claim, observe that $\lambda(C)$ can contain non-ground 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 KB and $\mathsf{FF}(KB)$ are $\mathbf{D}$-equisatisfiable.

Lemma 6.2.4. *KB is $\mathbf{D}$-unsatisfiable if and only if $\mathsf{FF}(KB)$ is $\mathbf{D}$-unsatisfiable.*

Proof. Since KB and Γ are $\mathbf{D}$-equisatisfiable by Lemma 6.2.1, the claim of the lemma can be demonstrated by showing that Γ and $\mathsf{FF}(KB)$ are $\mathbf{D}$-equisatisfiable. Furthermore, since Γ and $\mathsf{FF}(KB)$ are c-factored, by Lemma 5.1.6 it suffices to show that they are $\mathbf{d}$-equisatisfiable.

($\Leftarrow$) If $\mathsf{FF}(KB)$ is $\mathbf{d}$-unsatisfiable, since hyperresolution with superposition, concrete domain resolution and splitting, where all negative literals are selected, is sound and complete [9], a derivation of the empty clause from $\mathsf{FF}(KB)$ exists. We now show that such a derivation can be reduced to a derivation of the empty clause from Γ by sound inference rules, in particular, hyperresolution, paramodulation, instantiation, splitting and concrete domain resolution. In $\mathsf{FF}(KB)$, all clauses are safe, so electrons are always ground clauses, and each hyperresolvent is a ground clause. Furthermore, since superposition into variables is not necessary for completeness, positive and negative superposition inferences are performed only between ground clauses; the only exception is superposition into the first position of a clause $\neg T(a, y^{\mathsf{c}}) \vee y^{\mathsf{c}} \not\approx_{\mathbf{D}} b^{\mathsf{c}}$ obtained by c-factoring of $\neg T(a, b^{\mathsf{c}})$. Finally, splitting ground clauses simplifies the proof, since all ground clauses on a branch are unit clauses.

[2] Notice that λ is injective, but not surjective, so the definition of λ^{-} is correct.

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 non-equality predicate are selected. We show now by induction on n that, for any branch B, there exists a corresponding branch B' in a derivation from Γ by sound inference steps, and a set of clauses N'_m on B' such that: (*) if C is some 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 Γ 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 proposition (*) holds for some n and consider all possible inferences from premises in N_n deriving a clause C in $N_{n+1} = N_n \cup \{C\}$:

- A superposition into a literal $HU(u)$ is redundant, since HU is instantiated for each constant occurring in $\mathsf{FF}(KB)$, so the conclusion is already on the branch.

- Assume that the inference is a superposition from $s \approx t$ into the ground unit clause L. If L is of the form $S_f(u, v)$, 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 superposition of $u \approx v$ into $\neg T(u, y^{\mathsf{c}}) \vee y^{\mathsf{c}} \not\approx_{\mathbf{D}} b^{\mathsf{c}}$.

- 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'. Hence, reflexivity resolution can be applied on B' as well to derive the required counterpart clause. For non-ground clauses reflexivity resolution is not applicable, 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 inference with a nucleus C, a set of positive ground electrons $E_1, \ldots, E_k$ and a unifier σ, resulting in a hyperresolvent H. The clause C is safe, and by Corollary 6.2.3 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, H is a ground clause. Let σ' be the substitution obtained from σ 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' may 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 $f(x)$, 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'. Observe that 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 Δ_1 (Δ_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 Δ_1 or Δ_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 Δ'_1 and Δ'_2. We now perform superposition with equalities from Δ'_1 into C' at p'_x in the reverse order. After this, p'_x will contain the constant a, and p' will contain the term $f(a)$. Hence, we can apply superposition with equalities from Δ'_2 at p' in the original order. After this is done, each position p' will contain the term $\lambda^-(v)$. Let C'' denote the result of removing discrepancies at all positions. Obviously, $C'' = \lambda^-(C\sigma)$. All electrons 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 electrons with C'' to obtain H'. Obviously, $H' = \lambda^-(H)$, so the counterpart clause is derivable on B'.

- Since non-ground 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 branch B are unit clauses, and no clause in $\mathsf{FF}(KB)$ contains a positive literal with S_f or HU predicates, a ground non-unit 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 Γ as well.

($\Rightarrow$) If Γ is $\mathbf{d}$-unsatisfiable, since $\mathcal{BS}_{DL}^{\mathbf{D},+}$ is sound and complete, a derivation of the empty clause from Γ 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: (**) if C' is some clause in N'_n, then N_m contains the counterpart clause $C = \lambda(C')$. The induction base $n = 0$ is trivial. Assume now that (**) holds for some n and 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 σ' denote the unifier that the inference is performed with. By Corollary 5.2.2, σ' 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 λ 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 4.4.2, so the property (**) holds.

Now it is obvious that, if there is a derivation of the empty clause from Γ, then there is a derivation of the empty clause from $\mathsf{FF}(KB)$ as well. □

The result above means that $KB \models \alpha$ if and only if $\mathsf{FF}(KB) \models \alpha$, where α 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 the fact that, in checking $\mathbf{D}$-satisfiability of $\mathsf{FF}(KB)$, it is not necessary to perform a superposition inference into a literal of the form $HU(a)$.

6.3 Removing Irrelevant Clauses

The saturation of $\Gamma_{\mathcal{TR}g}$ derives new clauses which enable the reduction to $\mathsf{FF}(KB)$. However, the same process introduces many clauses which are not necessary. Consider, for example, the knowledge base $KB = \{A \sqsubseteq C, C \sqsubseteq B\}$. If the precedence of the predicate symbols is $C >_P B >_P A$, the saturation process will derive the clause $\neg A(x) \vee B(x)$, which is not necessary: all ground consequences of this clause can be obtained from clauses $\neg A(x) \vee C(x)$ and $\neg C(x) \vee B(x)$ only. Hence, we present an optimization, by which we reduce the number of clauses in the resulting disjunctive datalog program.

Definition 6.3.1. *For $N \subseteq \mathsf{FF}(KB)$, let $C \in N$ be a clause such that $\lambda^-(C)$ was derived in the saturation of $\Gamma_{\mathcal{TR}g}$ from premises P_i, $1 \leq i \leq k$, by an inference with a substitution σ. Then C is* irrelevant *w.r.t. N if $\lambda^-(C)$ is not derived by the decomposition rule and, for each premise P_i, $\lambda(P_i) \in N$ and 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 the sequence of clauses $\lambda^-(C_n), \ldots, \lambda^-(C_2), \lambda^-(C_1)$ corresponds to the order in which clauses are derived in 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 $\mathsf{FF_R}(KB) = N_n$ is called the relevant subset *of $\mathsf{FF}(KB)$.*

Removing irrelevant clauses preserves satisfiability, as demonstrated by the following lemma.

Lemma 6.3.2. *$\mathsf{FF_R}(KB)$ 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 ξ with a substitution σ, and $\lambda(P_i) \in N$, $i \leq i \leq k$. 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 $\mathbf{D}$-unsatisfiable if and only if some finite set M of ground instances of N is $\mathbf{D}$-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 τ, let $\lambda(P_i)\lambda(\sigma)\tau \in M'$, $1 \leq i \leq k$.

Let τ 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 σ 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 τ 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, it is easy to see that $\lambda(P_i)\lambda(\sigma)\tau \subseteq \lambda(P_i\sigma)\tau$. If the inclusion is strict, this is due to literals of the form $\neg S_f(a,b)$ in the latter clause, which do not occur in the first one because σ instantiates some variable from P_i to a functional 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 ξ, so D can be derived from M'. Hence, if M is $\mathbf{D}$-unsatisfiable, the set M' is $\mathbf{D}$-unsatisfiable as well. Since M' is a finite $\mathbf{D}$-unsatisfiable set of ground instances of $N \setminus \{C\}$, $N \setminus \{C\}$ is $\mathbf{D}$-unsatisfiable by Herbrand's theorem, so the property (*) holds.

Consider the sequence of clause sets $\mathsf{FF}(KB) = N_0, N_1, \ldots, N_n = \mathsf{FF_R}(KB)$ from Definition 6.3.1. For each $N_i = N_{i-1} \setminus \{C_i\}$, $i \geq 1$, the preconditions of property (*) are fulfilled, so by (*), N_i is $\mathbf{D}$-satisfiable if and only if N_{i-1} is $\mathbf{D}$-satisfiable. The claim of the lemma now follows by a straightforward induction on i. □

6.4 Reduction to Disjunctive Datalog

Reduction of an $\mathcal{ALCHIQ}(\mathbf{D})$ knowledge base KB to a disjunctive datalog program is now easy.

Definition 6.4.1. *For an $\mathcal{ALCHIQ}(\mathbf{D})$ knowledge base KB, the disjunctive datalog program $\mathsf{DD}(KB)$ is obtained by rewriting each clause $A_1 \vee \ldots \vee A_n \vee \neg B_1 \vee \ldots \vee \neg B_m$ from $\mathsf{FF_R}(KB)$ as the rule $A_1 \vee \ldots \vee A_n \leftarrow B_1, \ldots, B_m$.*

Theorem 6.4.2. *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 α 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 non-atomic 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 rules in* $\mathsf{DD}(KB)$ *is at most exponential, the number of literals in each rule is at most polynomial, 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 is an obvious consequence of Lemma 6.3.2. 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 $KB \cup \{\neg Q(a), \neg Q \sqsubseteq \neg C\} = KB \cup \{\neg Q(a), C \sqsubseteq Q\}$ is $\mathbf{D}$-unsatisfiable. Now the third claim follows from the second one, and the fact that Q is atomic.

By Lemma 4.3.8, $|\mathsf{Sat}(\Gamma_{\mathcal{TR}g})|$ is at most exponential in $|KB|$ and, for each clause $C \in \mathsf{Sat}(\Gamma_{\mathcal{TR}g})$, the number of literals in C is at most polynomial in $|KB|$. It is easy to see that the application of λ 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 unary coded, both c and f are polynomial in $|KB|$, so the number of constants a_f is also polynomial in $|KB|$. By Theorem 5.2.4, $\mathsf{Sat}(\Gamma_{\mathcal{TR}g})$ can be computed in time at most exponential in $|KB|$, so the fourth claim follows. □

6.5 Answering Queries in $\mathsf{DD}(KB)$

We discuss briefly the techniques for answering queries in $\mathsf{DD}(KB)$. Many techniques have been developed for disjunctive datalog without equality. These techniques can be used, provided that the usual congruence properties of equality are axiomatized correctly. This can be done by adding the following axioms to $\mathsf{DD}(KB)$, where the last axiom is instantiated for each predicate occurring in $\mathsf{DD}(KB)$ other than HU [30]:

$$x \approx x \leftarrow HU(x). \tag{6.1}$$

$$x \approx y \leftarrow y \approx x. \tag{6.2}$$

$$x \approx z \leftarrow x \approx y, y \approx z. \tag{6.3}$$

$$P(\ldots, y, \ldots) \leftarrow P(\ldots, x, \ldots), x \approx y. \tag{6.4}$$

Currently, the state-of-the-art technique for reasoning in disjunctive datalog is so-called *intelligent grounding* [27]. The algorithm is based on model building, which is performed by generating the ground instantiation of the program rules, generating candidate models, and eliminating those candidates which 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 which have exactly the same set of the stable models as the original program. This technique has been successfully implemented in the DLV disjunctive datalog engine [27].

Model building is of central interest in many disjunctive datalog applications. 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 this provides answering of ground queries only; non-ground ground queries are typically answered by considering all ground instances of the query. To reduce the number of relevant ground instances, the disjunctive datalog community is currently developing advanced heuristics. Furthermore, the algorithm computes the grounded program, which can be quite large even for small non-ground 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. Hence, we propose answering non-ground queries in $\mathsf{DD}(KB)$ by hyperresolution and basic superposition, which may be viewed as an extension of the fixpoint computation of plain datalog. This algorithm computes the set of all answers to a non-ground 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 non-ground program directly. Finally, the algorithm exhibits optimal worst-case complexity. A similar technique was presented in [19]; however, the algorithm presented there has two drawbacks: it does not take equality into account, and it does not specify whether application of redundancy elimination techniques is allowed.

Roughly speaking, our technique consists of saturating the rules and facts of $\mathsf{DD}(KB)$ by hyperresolution with basic superposition under an ordering in which all ground query literals are smallest. Additionally, it is required that the query predicate does not occur in the body of any rule. Under these assumptions, one may show that the saturated set of clauses contains all ground query literals cautiously entailed by the program. Because the ordering is total, in each ground disjunction there is exactly one maximal literal. Hence, the semi-naïve bottom-up computation or the join order optimizations can be adapted to the disjunctive case.

Definition 6.5.1. *For a predicate symbol Q, let $\mathcal{BS}^{\mathbf{D}}_{Q}$ denote the $\mathcal{BS}^{\mathbf{D}}$ calculus parameterized in the following way:*

- *All ground atoms of the form $Q(\mathbf{a})$ are smallest in the $\mathcal{E}$-term ordering $\succ$.*
- *All negative literals are selected.*

Furthermore, for any two closures $s \approx t \vee C$ where $s \approx t$ is strictly maximal with respect to C and $s \succ t$, and $Q(\mathbf{a}) \vee D$ where $Q(\mathbf{a})$ is strictly maximal with respect to D, $\mathcal{BS}^{\mathbf{D}}_{Q}$ performs any possible superposition from t into $Q(\mathbf{a})$, even if the corresponding position in $Q(\mathbf{a})$ is marked.

A remark about the first condition of the above definition is in order. Namely, any admissible $\mathcal{E}$-term ordering is stable under contexts and under substitutions, and it is total on ground terms; therefore, it has the subterm property for ground $\mathcal{E}$-terms [6]. However, such an ordering does not fulfill the first condition from the above definition: for literals $a \approx b$ and $Q(a)$ with $a \succ b$, we always have $Q(a) \succ a$, so $Q(a) \succ a \approx b$.

This situation can be remedied by dropping the requirement on $\succ$ to be stable under substitutions. Hence, for ground $\mathcal{E}$-terms the $\mathcal{E}$-term ordering $\succ$ must be total, well-founded and stable under contexts (i.e. for all ground $\mathcal{E}$-terms s, t and u, and all positions p, $s \succ t$ implies $u[s]_p \succ u[t]_p$). An example is a *query* ordering $\succ_Q$ induced over a total precedence of over constant symbols $>_C$ and predicate symbols $>_P$ such that Q is the smallest element in $>_P$, defined in the following way ($a_{(i)}$ and $b_{(i)}$ are arbitrary constants, P and R are arbitrary predicate symbols, and Q is the query predicate symbol):

- $a \succ_Q b$ if $a >_C b$,
- $P(a_1, \ldots, a_n) \succ_Q b$,
- $P(a_1, \ldots, a_n) \succ_Q R(b_1, \ldots, b_m)$ if $R >_P R$,
- $P(a_1, \ldots, a_n) \succ_Q P(b_1, \ldots, b_n)$ if there is some k, $1 \leq k \leq n$, such that $a_i = b_i$ for $i < k$ and $a_k \succ b_k$,
- $a \succ_Q Q(b_1, \ldots, b_n)$.

It is easy to see that $\succ_Q$ is well-founded, stable under contexts, and that it fulfills the requirements of Definition 6.5.1. However, it is not defined for non-ground $\mathcal{E}$-terms, since extending $\succ_Q$ to non-ground $\mathcal{E}$-terms would require $x \succ Q(x)$. Therefore, $\succ_Q$ cannot be used to decide satisfiability of a general first-order theory by $\mathcal{BS}^{\mathbf{D}}$.

However, $\mathbf{D}$-satisfiability of a positive program P can be decided by saturating P under $\mathcal{BS}^{\mathbf{D}}_Q$. Let P_∞ be the set of closures obtained by saturating P under $\mathcal{BS}^{\mathbf{D}}_Q$ and consider applying model-generation method to P_∞. All non-ground closures in P_∞ have selected negative literals, so they are not productive. Therefore, all productive clauses in P_∞ are positive ground clauses without functional terms. Literals from such clauses can be compared using $\succ_Q$, and a model can be generated in the same way as in [14, 65]. In fact, to saturate P and to generate a model of P_∞ it is not necessary to compare non-ground $\mathcal{E}$-terms, so stability under substitutions is not needed. Therefore, we conclude that $\mathcal{BS}^{\mathbf{D}}_Q$ is sound and complete for deciding satisfiability of P. From this we obtain the following result:

Lemma 6.5.2. *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 closures obtained by saturating the* c*-factor of P under $\mathcal{BS}^{\mathbf{D}}_Q$ up to redundancy.*

Proof. $P \models_c Q(\mathbf{a})$ if and only if the set of closures N', obtained as the result of saturating $P' \cup \{\neg Q(\mathbf{a})\}$ by $\mathcal{BS}^{\mathbf{D}}_Q$ up to redundancy contains the empty closure, where P' is a c-factor of P. Notice that, since all closures in P' are safe, all hyperresolvents are positive ground closures.

Consider first the case when no superposition inference is applied to the literal $\neg Q(\mathbf{a})$ in the saturation of N'. Since P is satisfiable, N' contains the empty closure if and only if a hyperresolution with $\neg Q(\mathbf{a})$ is performed in saturation. Since the literals containing Q are smallest in the ordering, a positive literal $Q(\mathbf{a})$ can be maximal only in a closure $C = Q(\mathbf{a}) \vee D$, where D contains only literals with the Q predicate. Since $\neg Q(\mathbf{a})$ is the only closure where Q occurs negatively, if D is not empty, no literal from D can be eliminated by a subsequent hyperresolution inference. Hence, the empty closure can be derived from such C if and only if D is empty, which is the case if and only if $Q(\mathbf{a}) \in N$.

Assume now that, in the saturation deriving N', several negative superposition inferences from closures $a_i \approx b_i \vee C_i$, $a_i \succ b_i$, are applied to $\neg Q(\mathbf{a})$, resulting in a closure $\neg Q(\mathbf{b}) \vee C$, which then is resolved with a closure $Q(\mathbf{b}) \vee D$, producing $C \vee D$. Such a derivation can be transformed into a derivation where superposition inferences are performed on b_i into $Q(\mathbf{b}) \vee D$, yielding $Q(\mathbf{a}) \vee C \vee D$, which can then participate in a resolution with $\neg Q(\mathbf{a})$ to obtain $C \vee D$. Thus, we may successively eliminate each superposition into some $\neg Q(\mathbf{a})$ and obtain a derivation in which no superposition into

$\neg Q(\mathbf{a})$ has been performed. Since in saturating N, all superposition inferences from the smaller side of the equality are performed into all literals containing Q, and all such inferences are sound, $Q(\mathbf{a}) \in N$, so the claim of the lemma follows. □

Assuming that Q is a single predicate, or that it 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 $Q(\mathbf{x}) \leftarrow \mathbf{A}(\mathbf{x})$ to satisfy the conditions of Lemma 6.5.2. We now consider the complexity of query answering in $\mathsf{DD}(KB)$.

Theorem 6.5.3. *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 computing the set of all ground literals of the form $C(a)$ or $R(a,b)$ for R an abstract role, entailed by $\mathsf{DD}(KB)$, can be done in time exponential in $|KB|$, assuming a bound on the arity of concrete predicates and for unary coding of numbers in input.*

Proof. In a way similar to Lemma 4.3.8, it is easy to see that the maximal length of each ground clause obtained in the saturation of $\mathsf{DD}(KB)$ by $\mathcal{BS}_Q^\mathbf{D}$ 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, which is polynomial in $|KB|$, giving rise to exponentially many different hyperresolution inferences. Hence, the saturation of $\mathsf{DD}(KB)$ may be performed in time exponential in $|KB|$. Since by Lemma 6.5.2 saturation of $\mathsf{DD}(KB)$ computes all certain answers of $\mathsf{DD}(KB)$, the claim of the theorem follows. □

Notice that the proof Theorem 6.5.3 immediately shows that the algorithm requires exponential space in the worst case. This is the main drawback to intelligent grounding, which requires only polynomial space.

An optimization of $\mathcal{BS}_Q^\mathbf{D}$ is applicable if KB employs the unique names assumption: assuming an ordering where individual constants a are smaller than constants b_f, superposition inferences from the smaller side of an equation are not needed. Namely, since $b_f \succ a$, no superposition inference may replace some a with b_f in a saturation of $\mathsf{DD}(KB) \cup \{\neg Q(\mathbf{a})\}$. Furthermore, for any clause $D = a \approx b \vee C$, there is a clause $a \not\approx b$, so D may be replaced immediately with C. Hence, no equality can reduce a position in $\neg Q(\mathbf{a})$, so we always have the first case from the proof of Lemma 6.5.2.

We briefly discuss the restriction of Theorem 6.5.3 requiring that R is an abstract role. Consider the knowledge base KB containing the only 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:

$$T(x,x_f) \leftarrow Q_1(x), S_f(x,x_f) \tag{6.5}$$
$$=_5(x_f) \leftarrow Q_1(x), S_f(x,x_f) \tag{6.6}$$
$$Q_1(a) \tag{6.7}$$
$$S_f(a,a_f) \tag{6.8}$$

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 closures, we apply c-factoring and obtain (6.9):

$$x \not\approx_{\mathbf{D}} 5 \leftarrow T(a,x) \tag{6.9}$$

We now saturate P under $\mathcal{BS}^{\mathbf{D}}$ by selecting all literals in the rule body; we thus obtain the well-known bottom-up hyperresolution strategy. The clause (6.10) is obtained by resolving (6.5) with (6.7) and (6.8); (6.11) is obtained by resolving (6.9) and (6.10); and (6.12) is obtained by resolving (6.6) with (6.7) and (6.8). Now the empty clause is derived from (6.11) and (6.12), since the set $S = \{a_f \not\approx_{\mathbf{D}} 5, =_5(a_f)\}$ is a $\mathbf{D}$-constraint.

$$T([a]\,,[a_f]) \tag{6.10}$$
$$[a_f] \not\approx_{\mathbf{D}} 5 \tag{6.11}$$
$$=_5([a_f]) \tag{6.12}$$
$$\square \tag{6.13}$$

Notice that, the program $\mathsf{DD}(KB)$ does not contain the constant 5; in the above refutation, the constant 5 is added to $\mathsf{DD}(KB)$ only by assuming the negation of the goal $T(a,5)$. Since $\mathsf{DD}(KB)$ does not contain the constant 5, the fact $T(a,5)$ is not derived by saturating $\mathsf{DD}(KB)$ by $\mathcal{BS}^{\mathbf{D}}_{Q}$. To summarize, we can freely use $\mathsf{DD}(KB)$ to answer queries by refutation, but we cannot use it in conjunction with the algorithm from Lemma 6.5.2 to answer queries regarding concrete roles or concrete literals.

6.6 Discussion

Complexity. The result of Theorem 6.5.3 may come as a surprise when compared to results from [26]. Namely, the complexity of cautious query answering in a non-ground positive disjunctive datalog program P is co-NExpTime, 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 predicate arity 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 propositional program $P^g = \mathsf{ground}(P)$, by replacing in each rule at most v variables with one of the c constants. Thus gives c^v combinations, so $|P^g|$ is exponential in $|P|$. Furthermore, the number of propositional symbols in P^g is bounded by $p \cdot c^a$, since in a ground atom each of the c constants can occur at each of the a positions, so the number of propositional symbols of P^g is also exponential in $|P|$. 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 propositional symbol of P^g; this can obviously be performed in non-deterministic exponential time 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 be performed in deterministic exponential time in $|P|$. These two steps combined give NExpTime complexity for satisfiability checking, which gives co-NExpTime complexity for unsatisfiability checking and query answering.

Note that the results in [27] 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 any model; one additionally needs to check whether 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.

Now co-NExpTime seems to be at odds with Theorem 6.5.3. However, the above calculation overestimates the complexity. Namely, query answering in $\mathsf{DD}(KB)$ can be reduced to unsatisfiability in the usual way. Satisfiability of $\mathsf{DD}(KB)$ can also be decided by computing $P^g_{DL} = \mathsf{ground}(\mathsf{DD}(KB))$. Let r be the number of rules, c the number of constants, and v the maximal number of variables in $\mathsf{DD}(KB)$; by Theorem 6.4.2, r is exponential, and c and v are polynomial in $|KB|$. Hence, $|P^g_{DL}|$ is bounded by $r \cdot c^v$, which is exponential in $|KB|$. However, an important difference to the previous case is the number of propositional symbols of P^g_{DL}: since predicates in $\mathsf{DD}(KB)$ are of bounded arity, the number of propositional symbols in P^g_{DL} is polynomial in $|KB|$. An interpretation I_{DL} for P^g_{DL} can be generated by choosing a truth value for each propositional symbols of P^g_{DL}. This can obviously be done in non-deterministic polynomial time in $|KB|$, so all interpretations can be examined in deterministic exponential time in $|KB|$. Finally, checking if I_{DL} is a model of P^g_{DL} can be performed in time exponential in $|KB|$, which gives the ExpTime complexity for satisfiability checking, and therefore 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 rules in $\mathsf{DD}(KB)$ is polynomial in $|KB|$, and (ii) the arity of predicates in $\mathsf{DD}(KB)$ is bounded.

Descriptive vs. Minimal-model Semantics. It is important to note that our reduction to disjunctive datalog preserves entailment under *descriptive semantics*. Namely, in [64] Nebel has shown that knowledge bases containing terminological cycles are not definitorial. This means that, for some fixed partial interpretation of atomic concepts, several interpretations of non-atomic 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, in [64] it is argued 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 so-called descriptive semantics, which coincides with that of Definition 3.2.4.

Obviously, our decision procedure implements exactly the descriptive semantics. Furthermore, Theorem 6.4.2 shows that $\mathsf{DD}(KB)$ entails exactly those ground facts which are entailed by our decision procedure, so $\mathsf{DD}(KB)$ also implements the descriptive semantics. Hence, one may say that the set of facts contained in any minimal model of $\mathsf{DD}(KB)$ coincides with the set of facts entailed by KB under descriptive semantics. Intuitively, the saturation process is responsible for this behavior.

Role of Basic Superposition. We point out that basic superposition is crucial for the correctness of the reduction. Namely, the basicness restriction renders superposition into Skolem function symbols redundant, which allows treating ground functional terms as constants.

Benefits of Reducing DLs to Disjunctive Datalog. Since we give separate complexity results and propose alternative query answering techniques, one may wonder whether the effort in reducing a description logic knowledge base to a disjunctive datalog program is justified. 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 7, we show that the rules can simply be appended to the result of the reduction; a result that is conceptually easy to grasp.

On the practical side, the reduction to disjunctive datalog has the benefit that it enables the application of the magic sets transformation [35]. This technique is independent of the query answering algorithm and can be reused as-is. Furthermore, if the program is not disjunctive, then we can answer queries using the standard fixpoint saturation, which has efficiently been implemented in practice.

6.7 Related Work

Our work was largely motivated by [36], where a decidable intersection of description logic and logic programming was investigated. In particular, the authors concentrate on 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 [88] to support more expressive description logic, provided that the rule engine supports advanced features. For example, if the rule engine supports equality, then functionality restrictions can be expressed in the knowledge base and executed by the translation. However, our approach is a significant extension, since we handle full $\mathcal{SHIQ}(\mathbf{D})$.

In [39] it was show how to convert $\mathcal{SHIQ}^*$ knowledge bases into so-called *conceptual logic programs* (CLP). The CLPs generalize the good properties of description logic to the answer set programming setting. 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 and therefore 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 is 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, Herbrand's universe of the programs obtained by the reduction is infinite, so existing answer set solvers cannot be used for decidable reasoning. In fact, decidability of reasoning is not considered at all.

The query answering algorithm from Section 6.5 was inspired by [19], 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 [19] by showing that simplification techniques can be applied in the fixpoint computation, and by extending the calculus to handle equality.

As already discussed in Section 6.5, the state-of-the-art technique for reasoning in disjunctive datalog is intelligent grounding [27]. It is actually a model building technique. Ground query answering problems are reduced to model building, and non-ground query answering is solved by refuting of confirming each possible ground query answer. Our query answering algorithm from Section 6.5 is not based on model building. It does not require grounding the disjunctive program, and it computes all answers to non-ground queries in one pass. However, it requires exponential space in the worst case, whereas intelligent ground requires only polynomial space.

7 Hybrid Reasoning by DL-safe Rules

In DIP deliverable D1.3 [61], the formalism of *DL-safe* rules was introduced to enable hybrid reasoning between description logics and other formalisms. Please refer to [61] for an in-depth discussion about the formalism and various aspects related to its semantics. In a nutshell, DL-safe rules allow integration of description logics with any other formalism which can be reduced to the formalism of function-free Horn clauses.

In [61] we have proven that the combination of DL-safe rules with description logics yields a hybrid formalism for which the problem of query answering is decidable. Our proof is based on a non-deterministic reduction of the query answering problem to the problem of checking knowledge base satisfiability. Although such a reduction is sufficient to show decidability, it is bound to be hopelessly inefficient in practice, mainly due to a high amount of don't-know non-determinism.

Since our goal is to provide efficient hybrid reasoning, in this chapter we present an alternative algorithm for reasoning with DL-safe rules. This algorithm relies upon our reduction to disjunctive datalog from Chapter 6 (and thus indirectly on algorithms from Chapter 4 and Chapter 5). Although the formalism of DL-safe rules has been introduced in [61], to make this deliverable self-contained, in sections 7.1 and 7.2 we repeat the necessary definitions, and then give the reasoning algorithm in Section 7.3.

7.1 Combining Description Logics and Rules

We now formalize the interface between $\mathcal{SHIQ}(\mathbf{D})$ and rules.

Definition 7.1.1 (DL Rules)**.** *Let KB be a $\mathcal{SHIQ}(\mathbf{D})$ knowledge base and let N_P be the set of predicate symbols such that $\{\approx\} \cup N_C \cup N_{R_a} \cup N_{R_c} \subseteq N_P$. For s and t constants or variables, a* DL-atom *is an atom of the form $A(s)$, where $A \in N_C$, of the form $R(s,t)$, where $R \in N_{R_a} \cup N_{R_c}$ and it is simple in KB, or of the form $s \approx t$. A* non-DL-atom *is an atom with a predicate from $N_P \setminus (N_C \cup N_{R_a} \cup N_{R_c} \cup \{\approx\})$. A (disjunctive)* DL-rule *is a (disjunctive) rule over N_P. A DL-program is a set of (disjunctive) DL-rules.*

The semantics of the combined knowledge base *(KB, P), where KB is a $\mathcal{SHIQ}(\mathbf{D})$ knowledge base and P is a DL-program, is given by translation into first-order logic as $\pi(KB) \cup P$, where each DL-rule $A_1 \vee ... \vee A_n \leftarrow B_1, ..., B_m$ is semantically equivalent to a clause $A_1 \vee ... \vee A_n \vee \neg B_1 \vee ... \vee \neg B_m$. The main inferences for (KB, P) are defined as follows:*

- Satisfiability checking. *(KB, P) is satisfiable if and only if $\pi(KB) \cup P$ is* **D**-*satisfiable.*
- Query answering. *A ground atom α is an* answer *of (KB, P), denoted with $(KB, P) \models \alpha$, if and only if $\pi(KB) \cup P \models_{\mathbf{D}} \alpha$.*

A few remarks regarding Definition 7.1.1 are in order.

Relationship with Existing Formalisms. The above definition yields a formalism compatible with the ones from [42, 54]. The main difference from [42] is that we allow non-DL-atoms to occur in a rule, and that we require only atomic concepts to occur in a rule. 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 [42], we immediately have that the reasoning with combined knowledge bases is undecidable. To achieve decidability, we introduce the notion of DL-safety in Section 7.2.

Minimal vs. First-order Models. Rules are usually interpreted under *minimal model* semantics, i.e. only models minimal w.r.t. set inclusion are considered; we write $P \models_c \alpha$ if a formula α is true in all minimal models of P. However, in Definition 7.2.1 we assume the standard first-order semantics for rules, where $P \models \alpha$ means that α is true in all models of P. We briefly discuss the differences between these two approaches and their practical consequences.

Assume that α is a positive ground atom. It is easy to see that in such a case, $P \models \alpha$ if and only if $P \models_c \alpha$. Namely, if α 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 α is a negative ground atom. In this case, there is a difference between minimal model semantics and first-order semantics, 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 perfectly valid first-order model of P and α 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 type of semantics also affects concept subsumption: let $\alpha = \forall x : C(x) \rightarrow D(x)$ and $P = \{C(a), D(a)\}$. Similarly as above, $P \not\models \alpha$; namely, $M_1 = \{C(a), D(a), C(b)\}$ is a model of P in which α is false. However, $M_2 = \{C(a), D(a)\}$ is the only minimal model of P and α is true in M_2, so $P \models_c \alpha$. The distinction between minimal models and general first-order models fundamentally changes the computational properties of query subsumption in disjunctive datalog: equivalence of general programs under minimal model semantics is undecidable [82], 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 query answering if queries are positive atoms; however, it is relevant for queries which involve negation or for concept subsumption. Negative queries are usually considered in a more general framework of *negation-as-failure*, where negation is interpreted as failure to prove a query, thus yielding a *non-monotonic* formalism. Whereas non-monotonic features are certainly very important for the Semantic Web, we do not address them here. Instead, our results are an initial step towards providing a practical hybrid knowledge representation formalism integrating description logics and rules. We also believe that our work may be used as a sound basis for future non-monotonic extensions.

7.2 DL-safety

We now introduce DL-safety restriction as one possible way to make reasoning with DL-rules decidable.

Definition 7.2.1 (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 may 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:

$$Homeworker(x) \leftarrow Person(x), livesAt(x, y), worksAt(x, y)$$

The reason for this is that both variables x and y occur in DL-atoms, but do not occur in a body atom with a predicate outside of KB. This 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 above rule becomes

$$Homeworker(x) \leftarrow Person(x), livesAt(x, y), worksAt(x, y), \mathcal{O}(x), \mathcal{O}(y), \mathcal{O}(z)$$

This rule is obviously DL-safe. We next discuss the consequences that this transformation has on the semantics.

7.3 Query Answering for DL-safe Rules

In a nutshell, we show that reasoning in (KB, P) can be performed by simply appending P to $\mathsf{DD}(KB)$. To show that, we modify the proofs of several lemmata leading to Theorem 6.4.2.

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 4.2. Observe that Definition 7.2.1 allows only simple roles to occur in DL-atoms, and that in Section 4.2 we show that the encoding preserves entailment of such atoms. Hence, for a DL-safe program P, it is easy to see that (KB, P) and $(\Omega(KB), P)$ are $\mathbf{D}$-equisatisfiable. Hence, in the rest we assume without loss of generality that KB is an $\mathcal{ALCHIQ}(\mathbf{D})$ knowledge base.

Let P^{c} be the c-factor of P. Without losing generality, we may assume that P^{c} is computed by first applying c-factoring to all negative DL-atoms, and then to non-DL-atoms in a rule. Thus, for a clause C from P^{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^{c} due to DL-safety occurs in a non-DL-atom, and in all positive DL-atoms containing a concrete role $T(x, y^{\mathsf{c}})$, due to DL-safety y^{c} also occurs in one negative non-DL-atom.

By Lemma 5.1.6, $\mathbf{D}$-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 are selected and nothing else is selected; if there are no negative non-DL-atoms, the selection function is the same as in Definition 4.3.3, i.e. it selects all negative binary literals.

We define the *extended* $\mathcal{ALCHIQ}(\mathbf{D})$-closures to include the closure types from Table 4.2 without conditions (iii) – (vi), the closure types from Table 5.2, and the closures corresponding to c-factors of DL-safe rules. Furthermore, closures of type 8 are allowed to contain positive or negative non-functional ground non-DL-atoms.

Lemma 7.3.1. *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 show the claim of the lemma, it is sufficient to show that the following property (*) holds: let $\Xi(KB) \cup P^{\mathsf{c}} = N_0, \ldots, N_i \cup \{C\}$ be a $\mathcal{BS}_{DL}^{\mathbf{D},+}$-derivation, where C is the conclusion derived from premises in N_i. Then C is either an 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-functional 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 some rule r. Since r is DL-safe, it can participate only in a hyperresolution inference with electrons 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 are variables z^{c} in literals of the form $\neg T(x, z^{\mathsf{c}})$ with T a concrete role. However, due to 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^{c} occur in non-DL-atoms, so in the conclusion these literals have 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 4.3.5, Lemma 5.2.1 and Theorem 4.4.8, so the property (*) holds. Since $\mathcal{BS}_{DL}^{\mathbf{D},+}$ is sound and complete for $\mathbf{d}$-satisfiability, the claim of the lemma follows. $\square$

The next step is to show that rules can simply be appended to the function-free version of KB.

Lemma 7.3.2. *(KB, P) is unsatisfiable if and only if $\mathsf{FF}(KB) \cup P^{\mathsf{c}}$ is $\mathbf{d}$-unsatisfiable.*

Proof. $\mathbf{d}$-satisfiability of $\mathsf{FF}(KB) \cup P^{\mathsf{c}}$ can be decided by a $\mathcal{BS}_{DL}^{\mathbf{D},+}$ saturation, in which all non-ground inferences are performed first. Since all non-DL-atoms in rules from P^{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 non-ground closures from $\Xi(KB) \cup P^{\mathsf{c}}$. Hence, as in Lemma 6.2.1, $\Xi(KB) \cup P^{\mathsf{c}}$ is $\mathbf{d}$-satisfiable if and only if $\Gamma = \mathsf{Sat}_{\mathsf{R}}(\Gamma_{\mathcal{TR}g}) \cup \Xi(KB_{\mathcal{A}}) \cup P^{\mathsf{c}}$ is $\mathbf{d}$-satisfiable.

It is now straightforward to extend Lemma 6.2.4 to show that Γ is $\mathbf{d}$-unsatisfiable if and only if $\mathsf{FF}(KB) \cup P$ is $\mathbf{d}$-unsatisfiable. For both directions of the proof, a hyperresolution with a rule r in one closure set can directly be simulated with a hyperresolution with r in the other closure set. $\square$

We now state our main result:

Theorem 7.3.3. *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*

$\mathbf{D}$*-unsatisfiable. Furthermore,* $(KB, P) \models \alpha$ *if and only if* $\mathsf{DD}(KB) \cup P \models_c \alpha$*, where* α *is a DL-atom* $A(a)$ *or* $R(a, b)$*, or* α *is a ground non-DL-atom.*

Proof. The first claim follows from Lemma 7.3.2 and the fact that $\mathbf{D}$-satisfiability of $\mathsf{FF}(KB) \cup P$ coincides with $\mathbf{d}$-satisfiability of $\mathsf{FF}(KB) \cup P^{\mathsf{c}}$. For the second claim, observe that $(KB, P) \models \alpha$ if and only if $(KB \cup \{\neg\alpha\}, P)$ is unsatisfiable. This is the case if and only if $\mathsf{FF}(KB \cup \{\neg\alpha\}) \cup P = \mathsf{FF}(KB) \cup P \cup \{\neg\alpha\}$ is $\mathbf{D}$-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 by using the algorithm from Section 6.5. We now determine its complexity.

Theorem 7.3.4. *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, and let* P *be a DL-safe program. Assuming unary coding of numbers and a bound on the arity of concrete domain predicates, computing all answers to a non-ground query in* (KB, P) *can be done 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. Similarly to Lemma 4.3.8, to determine the complexity of the algorithm we compute the maximal number of closures derived during saturation by $\mathcal{BS}_Q^{\mathbf{D}}$. 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 assumptions of the theorem, 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 closure is bounded by $\ell_1 = 2p(2c)^r$ (the first factor 2 allows each literal to occur positively or negatively, and the second factor 2 allows each term to be marked or not). If the predicate arity is bounded, then ℓ_1 is polynomial, and if the predicate arity is unbounded, it is exponential in $|KB| + |P|$. From Lemma 4.3.8 we know that the maximal number of DL-atoms in a closure, denoted as ℓ_2, is polynomial in $|KB|$ assuming a bound on the arity of concrete domain predicates and for unary coding of numbers. Since each ground closure can contain an arbitrary subset of these literals, the maximal number of ground closures derived by $\mathcal{BS}_Q^{\mathbf{D}}$ 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 $\mathsf{DD}(KB) \cup P$ can participate in a hyperresolution inference with ground closures in k^b ways, which is exponential in $|P|$. Furthermore, by Theorem 6.4.2 the number of rules t in $\mathsf{DD}(KB) \cup P$ 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 5.2.4, 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.

7.4 Related Work

$\mathcal{AL}$-log [24] combines a TBox and ABox expressed in the basic description logic $\mathcal{ALC}$ with datalog rules, which may be constrained with unary atoms having $\mathcal{ALC}$ concepts as predicates in the body. Query answering in $\mathcal{AL}$-log is decided by a variant of constrained resolution, combined with a tableaux algorithm for $\mathcal{ALC}$. The combined algorithm is shown to run in single non-deterministic exponential time. The fact that atoms with concept predicates can occur only as constraints in the body makes the rules 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 the effects of combining datalog rules with description logics is presented in [54]. The logic considered is $\mathcal{ALCNR}$, which, although less expressive than $\mathcal{SHIQ}$, contains constructors that are 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 non-recursive datalog rules, where both concepts and roles can occur in rule bodies, is also decidable, as 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 some non-DL-atom. As in $\mathcal{AL}$-log, query answering is decided using constrained resolution and a modified version of the tableaux calculus. Besides the fact that we treat 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) [42] 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 the rules, the formalism 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. DL-safe rules are a proper subset of SWRL, where some expressivity is traded for decidability. Hence, our approach provides an optimal query answering algorithm covering a significant portion of SWRL.

In [28] an approach for combining answer set programming with description logics was presented. The interaction between the subsystems is enabled by exchanging ground consequences between the two components. Hence, the consequences of the description logic knowledge base can be pushed as facts into the answer set program and vice versa. The set of derivable facts is obtained by fixpoint computation. In this approach, the two systems are not tightly integrated since interaction between the systems is performed only through the exchange of ground consequences.

The approaches from [36] and [88] for reducing certain fragments of description logics to logic programming can easily be extended with rules, by simply appending the rules to the result of the transformation. However, the description logic considered there does not support existential quantifiers, negation, or disjunction under positive polarity, so it is significantly less expressive than $\mathcal{SHIQ}(\mathbf{D})$. Hence, our approach is a proper extension.

8 Conclusion

In this deliverable we have presented the algorithms for hybrid reasoning, which will serve as a basis for the reasoning system delivered as part of Deliverable D1.7. The main goal of these algorithms is to integrate various logical formalisms in a unifying framework, and thus enable interoperability between them. Our main idea for hybrid reasoning is to reduce component formalisms to (positive disjunctive) datalog. In this way, a standard (disjunctive) datalog engine can be used to perform hybrid reasoning with any of the components.

Our algorithms are capable of handling a great number of formalisms used currently for ontology modeling. In particular, they can handle F-Logic, WSML, function-free Horn rules and OWL-DL (with the exception of nominals). Many of these formalisms can be translated into datalog rules in a relatively straightforward manner. For example, the translation of WSML ontologies into rules will be presented in Deliverable D2.7. However, translation of OWL-DL is technically involved, if decidability of the logic is to be preserved. Hence, in this deliverable we developed several algorithms which lead ultimately yield the given reduction. These algorithms are based on a saturation decision procedure by basic superposition [14, 65].

In our future work, we shall focus mainly on implementing these algorithms within Deliverable D1.7. Our ultimate goal is to produce a hybrid reasoning system which will serve as a foundation for reasoning with ontologies in DIP. We strongly believe that our algorithms are practicable, which we hope to confirm by the performance comparison, to be conducted as part of Deliverable D1.8. Our reasoning system will be used by a number of components in DIP, such as the repository for Web service descriptions, the service discovery component or the data mediation component.

References

[1] A. Abecker, D. Drollinger, and P. Hanschke. 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)*, pages 411–413. Springer, Kaiserslautern, Germany, 1991.

[2] G. Alsaç and C. Baral. Reasoning in description logics using declarative logic programming. Technical report, Arizona State University, Arizona, USA, 2002. 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*, 27:217–274, 1998.

[4] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook.* Cambridge University Press, January 2003.

[5] F. Baader and P. Hanschke. A Scheme for Integrating Concrete Domains into Concept Languages. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI-91)*, pages 452–457, Sydney, Australia, 1991.

[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. Kluwer Academic Publishers, Dordrecht, 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, November 1998.

[12] L. Bachmair and H. Ganzinger. Strict Basic Superposition. In *Automated Deduction—CADE-15*, volume 1421 of *Lecture Notes in Computer Science*, pages 160–174, Lindau, Germany, July 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 1st Int. Conf. on Logic Programming and Automated Reasoning (LPAR'92)*, volume 624, pages 119–130, St. Petersburg, Russia, July 1992. Springer.

[16] A. Borgida, R. J. Brachman, D. L. McGuinness, and L. A. Resnick. CLASSIC: a structural data model for objects. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 58–67, Portland, Oregon, United States, 1989. ACM Press.

[17] Alexander Borgida. On the Relative Expressiveness of Description Logics and Predicate Logics. *Artificial Intelligence*, 82(1-2):353–367, 1996.

[18] R. Boyer. *Locking: A Restriction of Resolution*. PhD thesis, University of Texas at Austin, Texas, USA, 1971.

[19] S. Brass and U. W. Lipeck. Generalized Bottom-Up Query Evaluation. In A. Pirotte, C. Delobel, and G. Gottlob, editors, *Advances in Database Technology - Proceedings EDBT'92*, pages 88–103, Berlin, 1992. Springer.

[20] C.-L. Chang and R. C.-T. Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, Inc., 1997.

[21] H. de Nivelle. A Resolution Decision Procedure for the Guarded Fragment. In *Proc. of the 15th Int. Conf. on Automated Deduction*, pages 191–204. Springer, 1998.

[22] 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 2001. Springer.

[23] 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.

[24] F. M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. AL-log: Integrating Datalog and Description Logics. *J. of Intelligent Information Systems*, 10(3):227–252, 1998.

[25] 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)*, pages 69–74. Springer, Ottenstein/Niederösterreich, September 1986.

[26] T. Eiter, G. Gottlob, and H. Mannila. Disjunctive Datalog. *ACM Transactions on Database Systems*, 22(3):364–418, 1997.

[27] 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, *Proceedings of the 4th International Conference on Logic Programming and Non-monotonic Reasoning (LPNMR'97)*, volume 1265 of *Lecture Notes in Artificial Intelligence*, pages 364–375, Dagstuhl, Germany, July 1997. Springer.

[28] T. Eiter, T. Lukasiewicz, R. Schindlauer, and H. Tompits. Combining Answer Set Programming with Description Logics for the Semantic Web. In *Proc. of the 9th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR2004)*. AAAI Press, 2004. To appear.

[29] C. Fermuller, T. Tammet, N. Zamov, and A. Leitsch. *Resolution Methods for the Decision Problem*. Springer, 1993.

[30] M. Fitting. *First-Order Logic and Automated Theorem Proving, 2nd Edition*. Springer, 1996.

[31] H. Ganzinger and H. de Nivelle. A superposition decision procedure for the guarded fragment with equality. In *Proceedings of the 14th IEEE Symposium on Logic in Computer Science*, pages 295–305. IEEE Computer Society Press, 1999.

[32] 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, *Advances in Modal Logic, Volume 2*, volume 119 of *Lecture Notes*, pages 225–246. CSLI Publications, Stanford, USA, 2001.

[33] G. Gottlob and A. Leitsch. On the efficiency of subsumption algorithms. *Journal of the ACM*, 32(2):280–295, 1985.

[34] E. Grädel, M. Otto, and E. Rosen. Two-Variable Logic with Counting is Decidable. In *Proc. of 12th IEEE Symposium on Logic in Computer Science LICS '97*, Warsaw, Poland, 1997.

[35] S. Greco. Binding Propagation Techniques for the Optimization of Bound Disjunctive Queries. *IEEE Transactions on Knowledge and Data Engineering*, 15(2):717–736, March/April 2003.

[36] B. N. Grosof, I. Horrocks, R. Volz, and S. Decker. Description Logic Programs: Combining Logic Programs with Description Logic. In *Proc. of the Twelfth Int. World Wide Web Conf. (WWW 2003)*, pages 48–57. ACM, 2003.

[37] V. Haarslev and R. Möller. RACER System Description. In *1st Int. Joint Conf. on Automated Reasoning (IJCAR-01)*, pages 701–706. Springer, 2001.

[38] V. Haarslev, R. Möller, and M. Wessel. The Description Logic ALCNHR+ Extended with Concrete Domains: A Practically Motivated Approach. In T. Nipkow R. Gore, A. Leitsch, editor, *Proc. of Int. Joint Conf. on Automated Reasoning, IJCAR 2001*, pages 29–44, Siena, Italy, June 18-23 2001. Springer.

[39] S. Heymans and D. Vermeir. Integrating Semantic Web Reasoning and Answer Set Programming. In M. De Vos and A. Provetti, editors, *Answer Set Programming, Advances in Theory and Implementation, Proc. of the 2nd Int. ASP'03 Workshop*, Volume 78 of CEUR Proceedings, pages 194–208, Messina, Sicily, September 2003.

[40] I. Hodkinson. Loosely guarded fragment of first-order logic has the finite model property. *Studia Logica*, 70:217–274, 2002.

[41] I. Horrocks. Using an Expressive Description Logic: FaCT or Fiction? In A. G. Cohn, L. Schubert, and S. C. Shapiro, editors, *Proc. 6th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 636–647, Trento, Italy, June 1998. Morgan Kaufmann Publishers.

[42] I. Horrocks and P. F. Patel-Schneider. A Proposal for an OWL Rules Language. In *Proc. of the Thirteenth Int. World Wide Web Conf.(WWW 2004)*. ACM, 2004.

[43] I. Horrocks and U. Sattler. Ontology Reasoning in the $\mathcal{SHOQ}(\mathbf{D})$ Description Logic. In B. Nebel, editor, *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*, pages 199–204. Morgan Kaufmann, 2001.

[44] I. Horrocks, U. Sattler, and S. Tobies. Practical Reasoning for Very Expressive Description Logics. *Logic Journal of the IGPL*, 8(3):239–263, 2000.

[45] U. Hustadt. *Resolution-Based Decision Procedures for Subclasses of First-Order Logic*. PhD thesis, Universität des Saarlandes, Saarbrücken, Germany, November 1999.

[46] 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 2004.

[47] U. Hustadt, B. Motik, and U. Sattler. Reducing $\mathcal{SHIQ}^-$ Description Logic to Disjunctive Datalog Programs. In D. Dubois, C. Welty, and M.-A. Williams, editors, *Proc. of the 9th Int. Conf. on Knowledge Representation and Reasoning (KR 2004)*, pages 152–162, Menlo Park, California, USA, June 2004. AAAI Press.

[48] U. Hustadt and R. A. Schmidt. Issues of Decidability for Description Logics in the Framework of Resolution. In R. Caferra and G. Salzer, editors, *Automated Deduction in Classical and Non-Classical Logics*, volume 1761 of *LNAI*, pages 192–206. Springer, 1999.

[49] W. H. Joyner Jr. Resolution Strategies as Decision Procedures. *Journal of the ACM*, 23(3):398–417, 1976.

[50] B. Kallick. A decision procedure based on the resolution method. In A. J. H. Morrell, editor, *Proc. of the IFIP Congress*, pages 269–275. Horth-Holland, 1968. Volume 1 - Mathematics, Software.

[51] Y. Kazakov and H. de Nivelle. A Resolution decision procedure for the guarded fragment with transitive guards. In *Proc. of 2nd Int. Joint Conference on Automated Reasoning (IJCAR 2004)*, Cork, Ireland, 2004. To appear.

[52] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42(4):741–843, 1995.

[53] A. Leitsch. Deciding clause classes by semantic clash resolution. *Fundamenta Informaticae*, 18:63–182, 1993.

[54] A. Y. Levy and M.-C. Rousset. Combining Horn rules and description logics in CARIN. *Artificial Intelligence*, 104(1-2):165–209, 1998.

[55] J. W. Lloyd. *Foundations of logic programming; (2nd extended ed.)*. Springer-Verlag, New York, 1987.

[56] B. Löchner. Things to know when implementing LPO. In *IJCAR Workshop on Empirically Successful First OrderReasoning (ESFOR)*, Electronic Notes in Theoretical Computer Science, 2004. To appear.

[57] C. Lutz. *The Complexity of Reasoning with Concrete Domains*. PhD thesis, Teaching and Research Area for Theoretical Computer Science, RWTH Aachen, 2002.

[58] C. Lutz. NExpTime-complete Description Logics with Concrete Domains. *ACM Transactions on Computational Logic*, 2003. To appear.

[59] 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, *Advances in Modal Logics*, volume 3. CSLI Publications, Stanford, 2001.

[60] William McCune. Solution of the Robbins Problem. *Journal of Automated Reasoning*, 19(3):263–276, 1997.

[61] B. Motik. D1.3: Framework for hybrid, modularized ontology representation and reasoning for semantics-based services, 2004.

[62] 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 *Lecture Notes in Computer Science*, pages 549–563, Hiroshima, Japan, 2004. Springer.

[63] B. Motik and S. Spaccapietra. D1.1: Report on the requirements analysis and the state of the art, 2004.

[64] 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.

[65] R. Nieuwenhuis and A. Rubio. Theorem Proving with Ordering and Equality Constrained Clauses. *Journal of Logic and Computation*, 19(4):312–351, April 1995.

[66] 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.

[67] H. De Nivelle, R. A. Schmidt, and U. Hustadt. Resolution-Based Methods for Modal Logics. *Logic Journal of the IGPL*, 8(3):265–292, May 2000.

[68] 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.

[69] J. Pan and I. Horrocks. Web Ontology Reasoning with Datatype Groups. In D. Fensel, K. Sycara, and J. Mylopoulos, editors, *Proc. of the 2nd International Semantic Web Conference (ISWC 2003)*, number 2870 in Lecture Notes in Computer Science, pages 47–63. Springer, 2003.

[70] J.Z. Pan and I. Horrocks. Extending Datatype Support in Web Ontology Reasoning. In *Proc. of the 2002 Int. Conf. on Ontologies, Databases and Applications of SEmantics (ODBASE 2002)*, volume 2519 of *Lecture Notes in Computer Science*, pages 1067–1081. Springer, 2002.

[71] P. F. Patel-Schneider, P. Hayes, and I. Horrocks. OWL Web Ontology Language; Semantics and Abstract Syntax. http://www.w3.org/TR/owl-semantics/, November 2002.

[72] N. Peltier. On the decidability of the PVD class with equality. *Logic Journal of the IGPL*, 9(4):601–624, 2001.

[73] D. A. Plaisted and S. Greenbaum. A Structure-preserving Clause Form Transformation. *Journal of Symbolic Logic and Computation*, 2(3):293–304, 1986.

[74] I. Pratt-Hartmann. Counting Quantifiers and the Stellar Fragment. Technical report, University of Manchester, UK, 2003. submitted for publishing.

[75] R. Reiter. Two Results on Ordering for Resolution with Merging and Linear Format. *Journal of the ACM*, 18(4):630–646, 1971.

[76] A. Riazanov and A. Voronkov. Splitting Without Backtracking. In B. Nebel, editor, *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*, pages 611–617, Seattle, WA, USA, August 2001. Morgan Kaufmann.

[77] J. A. Robinson. A Machine-Oriented Logic Based on the Resolution Principle. *Journal of the ACM*, 12(1):23–41, 1965.

[78] J. A. Robinson. Automatic deduction with hyper-resolution. *International Journal of Computational Mathematics*, 1(3):227–234, 1965.

[79] K. Schild. A correspondence theory for terminological logics: preliminary report. In *Proc. of 12th Int. Joint Conference on Artificial Intelligence (IJCAI '91)*, pages 466–471, Sidney, AU, 1991.

[80] 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 *Lecture Notes in Artificial Intelligence*, pages 433–448. Springer, June 17–20 2000.

[81] R. A. Schmidt and U. Hustadt. A Principle for Incorporating Axioms into the First-Order Translation of Modal Formulae. In F. Baader, editor, *Automated Deduction—CADE-19*, volume 2741 of *Lecture Notes in Artificial Intelligence*, pages 412–426. Springer, 2003.

[82] O. Shmueli. Equivalence of datalog queries is undecidable. *Journal of Logic Programming*, 15(3):231–241, 1993.

[83] W. Snyder. On the complexity of recursive path orderings. *Information Processing Letters*, 46(5):257–262, 1993.

[84] M. E. Stickel. Automated Deduction by Theory Resolution. *Journal of Automated Reasoning*, 1(4):333–355, 1985.

[85] T. Tammet. *Resolution Methods for Decision Problems and Finite-Model Building*. PhD thesis, Chalmers University of Technology / University of Göteborg, 1992.

[86] S. Tobies. *Complexity Results and Practical Algorithms for Logics in Knowledge Representation*. PhD thesis, RWTH Aachen, Germany, 2001.

[87] M. Vardi. Why is modal logic so robustly decidable? In N. Immerman and P. Kolaitis, editors, *Descriptive Complexity and Finite Models*, volume 31 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 149–184. AMS, 1997.

[88] R. Volz. *Web Ontology Reasoning With Logic Databases*. PhD thesis, Universität Fridericiana zu Karlsruhe (TH), Germany, 2004.