SATURATION-BASED DECISION PROCEDURES

FOR

EXTENSIONS OF THE GUARDED FRAGMENT

Dissertation

zur Erlangung des Grades

Doktor der Ingenieurwissenschaften (Dr.-Ing.)

der Naturwissenschaftlich-Technischen Fakultät I

der Universität des Saarlandes

von

Yevgeny Kazakov

Saarbrücken

2005

Verfasser:
Yevgeny Kazakov
Max-Planck Institut für Informatik
Stuhlsatzenhausweg 85
66123 Saarbrücken
Germany
ykazakov@mpi-inf.mpg.de

Dekan:
Prof. Dr. Jörg Eschmeier

Prüfungsausschuss:
Prof. Dr. Jörg Siekmann
Prof. Dr. Franz Baader
Prof. Dr. Gert Smolka
Dr. Hans de Nivelle
Dr. Uwe Waldman

Tag des Kolloquiums:
17.03.2006

ABSTRACT

We apply the framework of Bachmair and Ganzinger for saturation-based theorem proving to derive a range of decision procedures for logical formalisms, starting with a simple terminological language $\mathcal{EL}$, which allows for conjunction and existential restrictions only, and ending with extensions of the guarded fragment with equality, constants, functionality, number restrictions and compositional axioms of form $S \circ T \subseteq H$. Our procedures are derived in a uniform way using standard saturation-based calculi enhanced with simplification rules based on the general notion of redundancy. We argue that such decision procedures can be applied for reasoning in expressive description logics, where they have certain advantages over traditionally used tableau procedures, such as optimal worst-case complexity and direct correctness proofs.

ZUSAMMENFASSUNG

Wir wenden das Framework von Bachmair und Ganzinger für saturierungsbasiertes Theorembeweisen an, um eine Reihe von Entscheidungsverfahren für logische Formalismen abzuleiten, angefangen von einer simplen terminologischen Sprache $\mathcal{EL}$, die nur Konjunktionen und existentielle Restriktionen erlaubt, bis zu Erweiterungen des Guarded Fragment mit Gleichheit, Konstanten, Funktionalität, Zahlenrestriktionen und Kompositionsaxiomen der Form $S \circ T \subseteq H$. Unsere Verfaren sind einheitlich abgeleitet unter Benutzung herkömmlicher saturierungsbasierter Kalküle, verbessert durch Simplifikationsregeln, die auf dem Konzept der Redundanz basieren. Wir argumentieren, daß solche Entscheidungsprozeduren für das Beweisen in ausdrucksvollen Beschreibungslogiken angewendet werden können, wo sie gewisse Vorteile gegenüber traditionell benutzten Tableauverfahren besitzen, wie z.B. optimale worst-case Komplexität und direkte Korrektheitsbeweise.

Посвящается моим родителям

To my parents

Extended Abstract

Description logics (DLs) are families of languages used for representation of knowledge and conceptual modelling. In conceptual modelling one often needs to organise classes in hierarchies, which requires checking the subsumption relations between classes. These and other reasoning tasks can be solved by dedicated reasoning procedures which are specially designed for every description logic. Today the research in description logics is mainly concentrated on (1) the development of practical algorithms for DLs [Horrocks, Sattler & Tobies, 2000] and (2) study of computational complexity for reasoning problems in different DLs [see Donini, 2003; Tobies, 2001].

Most algorithms for description logics are based on so-called tableau procedures. These procedures are shown to behave well in practice, however they have several disadvantages: (***i***) such procedures typically exhibit suboptimal worst-case complexity, (***ii***) they rely on (some form of) a tree-model property for DL and (***iii***) they do not use the formal semantics of DL-constructors directly. These limitations might become a serious problem for development of reasoning algorithms for very expressive description logics: this problem already arises with the DL-based ontology language OWL for the semantic web [see Horrocks & Sattler, 2005].

In this thesis we propose an alternative approach for designing reasoning algorithms for description logics which avoids the above limitations. We argue that the framework of Bachmair & Ganzinger [1994, 1990] for saturation-based theorem proving can be used for engineering of decision procedures for many logical formalisms that can be translated to first-order logic.

It is well-known that description logics correspond to certain fragments of first-order logic via the standard semantical translation of DL-constructors. Hence general theorem proving calculi for first-order logics, such as ordered resolution or ordered paramodulation, can be applied for solving reasoning tasks in description logics. Such methods give only semi-decision procedures for first-order fragments, since they do not terminate in general. Joyner Jr. [1976] has noticed that for certain fragments of first-order logic, general theorem proving methods can be turned into decision procedures. His method has been later extended to a variety of other first-order fragments and, via translations, to non-classical logics [see Fermüller, Leitsch,

Hustadt & Tammet, 2001].

To ensure termination, such procedures typically use a collection of “tricks” – like non-liftable orders, renaming or hyperresolution – which work well for one fragment but do not apply for others. We demonstrate that many of these “tricks” can be formulated as optional simplification rules. In theorem provers simplification rules are used to prune the search space of a prover. We found that simplification rules can be also used to eliminate potentially dangerous clauses. For example, two clauses $a(x, x) \vee a(x, y)$ and $\neg a(x, x) \vee \neg a(x, z)$ may be resolved on their first literals producing a clause $a(x, y) \vee \neg a(x, z)$ with more variables. This inference can be blocked using a non-liftable order that makes the last literals in these clauses larger. The same effect can be achieved by splitting the input clauses in $a_1(x) \vee a(x, y)$, $\neg a_1(x) \vee a(x, x)$ and $\neg a_2(x) \vee \neg a(x, z)$, $a_2(x) \vee \neg a(x, x)$, and using liftable orders in which unary literals are smaller than binary ones. It is easy to see that the number of variables does not grow with resolution on binary literals. Hence the effect of non-liftable orders in this example is simulated with an additional simplification rule which splits the clauses.

Simplification rules can be justified by the general notion of redundancy, introduced by Bachmair & Ganzinger [1994, 1990], according to which a clause can be deleted if it is a consequence of smaller clauses. Simplification rules produce exactly these smaller clauses. The advantage of simplification rules is that they can be “plugged-in” to virtually every saturation procedure. This gives a simple recipe for engineering of decision procedures: apply a saturation strategy and resolve dangerous cases using simplification rules. To illustrate this approach, we give a uniform description of several well-known resolution-based decision procedures for the guarded, two-variable and monadic fragments without equality, and show how they can be combined in a modular way. Moreover, we prove that these procedures have the best known complexities.

We apply our framework to a range of first-order fragments that are related to description logics. First, we consider a simple terminological language $\mathcal{EL}$ which allows for conjunction and existential restrictions only. DL $\mathcal{EL}$ is robustly tractable: subsumption of concepts in $\mathcal{EL}$ w.r.t. (cyclic) terminologies is polynomially decidable [Baader, 2002] and remains so under many extensions [Baader, Brandt & Lutz, 2005]. Tractability of $\mathcal{EL}$ can be partially explained in that all clauses obtained from translating $\mathcal{EL}$-terminologies are Horn. However, this is not enough, since the Horn logic is undecidable. The second important property is that the Skolem functions that appear in such clauses, determine uniquely their non-functional part: it is not possible to have clauses $\neg A(x) \vee R(x, f(x))$ and $\neg B(x) \vee C(f(x))$ with $A \neq B$. This turns out to be an invariant under resolution inferences. These two properties insure that ordered resolution with appropriate selection produces only a cubic number of clauses. It also gives us a hint of how to extend $\mathcal{EL}$ preserving tractability.

A surprising property of saturation procedures for $\mathcal{EL}$ is that they can be implemented right away without using any theorem prover. This is possible, because all resolution inferences for $\mathcal{EL}$, can be encoded as datalog rules, for example:

$$\begin{array}{lccccc} \text{Inference:} & \neg A(x) \vee \underline{B(f(x))}; & \neg \underline{B(x)} \vee C(x) & \vdash & \neg A(x) \vee C(f(x)) \\ \text{Is encoded by:} & \mathsf{C}(A, B, f), & \mathsf{D}(B, C) & \rightarrow & \mathsf{C}(A, C, f). \end{array}$$

To obtain saturation-based procedures for more expressive description logics, we extend a decision procedure of Ganzinger & de Nivelle [1999] for the guarded fragment. The guarded fragment has been introduced by Andréka, van Benthem & Németi [1996] to transfer good properties of modal-like languages to first order formulas. It is obtained by restricting quantification in first-order formulas to the bounded form: $\forall \overline{y}.[G \rightarrow F]$ or $\exists \overline{y}.[G \wedge F]$, where G is an atom-guard containing all free variables of F. The guarded fragment captures many expressive description logics, like DL $\mathcal{ALCIH}$. However some constructors in description logics like nominals, functional roles and (qualified) number restrictions do not correspond to guarded formulas. For every of these constructors, we consider an appropriate extension of the guarded fragment.

Restrictions for nominals correspond to first-order formulas of form $\forall xy.[a(x) \wedge a(y) \rightarrow x \simeq y]$, which are not guarded. Nominals can be expressed alternatively by formula $\forall x.[a(x) \rightarrow x \simeq c_a]$, which is a guarded formula containing a constant c_a. Hence, to capture nominals it suffices to extend a saturation-based procedure to the guarded fragment with constants, which can be done in two ways: using elimination of constants described in [Grädel, 1999], or directly.

For functional restrictions and counting we are faced with a problem that guarded fragment with functionality is undecidable in general [Grädel, 1999]. Since roles in description logics are mostly used as guards, we consider a restriction of the guarded fragment where functional/number restrictions can be imposed on guards only. These fragments turned out to be decidable by paramodulation with special simplification rules.

Finally, we are concerned with extensions of the guarded fragment with compositional axioms of form $S \circ T \subseteq H$. Such extensions are known to be very problematic, since compositional axioms in many cases result in undecidability [Schmidt-Schauß, 1989; Grädel, 1999; Demri, 2001]. At the same time, integration of compositional axioms is challenging, because many relations are characterised by such axioms: temporal relations, orderings, distances and topological relations. We analyse a variety of extensions of the guarded fragment with compositional axioms and identify the border between “dangerous” and “safe” usage of such axioms. For our decidability proofs we employ a refined version of the ordered chaining calculus from [Bachmair & Ganzinger, 1998*b*, 1995] and a novel kind of simplification rules, like the following one for transitivity:

$$\texttt{TC}: \frac{\neg(xTy) \vee \neg a(x) \vee b(y)}{\begin{array}{r}\neg(xTy) \vee \neg a(x) \vee a^T(y) \\ \neg(xTy) \vee \neg a^T(x) \vee a^T(y) \\ \neg a^T(y) \vee b(y)\end{array}}$$

In contrast to conventional simplification rules, such rules do not make clauses redundant but they make inferences redundant: the above rule introduces a new atom $a^T(x)$ such that (1) $a^T(y)$ is T-reachable from $a(x)$ in one step, (2) $a^T(y)$ is closed under T, and (3) $a^T(y)$ is contained in $b(y)$. After applying this rule, we may "forget" that T is transitive in the involved clauses, which formally means that the chaining inferences with these clauses become redundant.

Ausführliche Zusammenfassung

Beschreibungslogiken (BL) sind Sprachfamilien, die für die Wissensrepräsentation und konzeptuelle Modellierung benutzt werden. In der konzeptuellen Modellierung müssen oft Klassen in Hierarchien organisiert werden, wozu es notwendig ist, die Subsumptionsbeziehungen zwischen Klassen zu berechnen. Diese und andere Beweisaufgaben können von spezialisierten Beweiser, die speziell für jede Beschreibungslogik entwickelt wurden, gelöst werden. Die Forschung auf dem Gebiet der Beschreibungslogiken konzentriert sich gegenwärtig hauptsächlich auf (1) Entwicklung praxisnaher Algorithmen für BL [Horrocks et al., 2000] und (2) Erforschung der Komplexität von Beweisaufgaben in unterschiedlichen BL [siehe Donini, 2003; Tobies, 2001].

Die meisten Algorithmen für Beschreibungslogiken basieren auf sogenannten Tableauverfahren. Diese Verfahren haben sich in der Praxis bewährt, jedoch sind sie mit einigen Nachteilen behaftet: (***i***) Solche Verfahren weisen suboptimale Worst-Case-Komplexität auf, (***ii***) sie benötigen eine Baummodelleigenschaft der Beschreibungslogik, (***iii***) sie benutzen nicht direkt die formale Semantik von BL-Konstruktoren. Diese Beschränkungen können ein ernsthaftes Problem für sehr ausdrucksstärke Beschreibungslogiken darstellen: Dieses Problem tritt bereits bei der BL-basierten Ontologiesprache OWL für das Semantische Web [siehe Horrocks & Sattler, 2005] auf.

In dieser Dissertation stellen wir einen alternativen Ansatz für das Design von Beweisalgorithmen für Beschreibungslogiken vor, welcher die obigen Probleme vermeidet. Wir werden argumentieren, daß das Framework für saturierungsbasiertes Theorembeweisen von Bachmair & Ganzinger [1994, 1990] für die Konstruktion von Entscheidungsverfahren für viele logische Formalismen benutzt werden kann, die in eine Logik erster Stufe übersetzt werden können.

Es ist wohlbekannt, daß BL gewissen Fragmenten der Logik erster Stufe mittels der standardmäßigen semantischen Übersetzung von BL-Konstruktoren entsprechen. Deshalb können allgemeine Theorembeweismethoden der Logik erster Stufe, z.B. geordnete Resolution oder geordnete Paramodulation, auf die Lösung von Beweisaufgaben in BL angewendet werden. Diese Methoden ermöglichen lediglich Semi-Entscheidungsverfahren für Fragmente erster Stufe, da sie im Allgemeinen

nicht terminieren. Joyner Jr. [1976] hat bemerkt, daß für gewisse Fragmente der Logik erster Stufe generelle Theorembeweismethoden in Entscheidungsverfahren umgewandelt werden können. Seine Methode wurde später ausgeweitet auf eine Vielzahl anderer Fragmente erster Stufe und durch Übersetzungen auf nicht-klassische Logiken [siehe Fermüller et al., 2001].

Um eine Terminierung sicherzustellen bedienen sich solche Verfahren typischerweise einer Sammlung von "Tricks" – wie nicht-liftbare Ordnungen, Umbenennungen oder Hyperresolution – die gut für ein gewisses Fragment funktionieren, aber nicht für andere. Wir zeigen, daß viele dieser Tricks als optionale Vereinfachungsregeln formuliert werden können. Bei Theorembeweisern werden Vereinfachungsregeln benutzt, um den Suchraum eines Beweisers einzuschränken. Wir haben herausgefunden, daß Vereinfachungsregeln benutzt werden können, um potentiell gefährliche Klauseln zu eliminieren. Z.B. kann es vorkommen daß zwei Klauseln $a(x,x) \vee a(x,y)$ und $\neg a(x,x) \vee \neg a(x,z)$ auf ihrem ersten Literal resolviert werden, wobei eine Klausel $a(x,y) \vee \neg a(x,z)$ mit mehr Variablen entsteht. Diese Inferenz kann vermieden werden, indem eine nicht-liftbare Ordnung benutzt wird, die die letzten Literale der Klauseln größer macht. Der gleiche Effekt kann durch die Teilung der Inputklauseln in $a_1(x) \vee a(x,y)$, $\neg a_1(x) \vee a(x,x)$ und $\neg a_1(x) \vee \neg a(x,z)$, $a_1(x) \vee \neg a(x,x)$ und der Benutzung von liftbaren Ordnungen, bei denen unäre Literale kleiner sind als binäre Literale, erreicht werden. Es ist leicht erkennbar, daß die Anzahl der Variablen durch Resolution auf binären Literalen nicht wächst. So wird in diesem Beispiel der Effekt von nicht-lifbaren Ordnungen simuliert durch eine zusätzliche Vereinfachungsregel, die die Klauseln teilt.

Vereinfachungsregeln lassen sich rechtfertigen durch das Prinzip der Redundanz, eingeführt von Bachmair & Ganzinger [1994, 1990], wonach eine Klausel gelöscht werden kann, wenn sie aus kleineren Klauseln folgt. Vereinfachungsregeln produzieren genau diese kleineren Klauseln. Der Vorteil von Vereinfachungsregeln ist, daß sie in quasi jede Saturierungsprozedur "eingefügt" werden können. Dies ergibt eine einfache Anleitung, um Entscheidungsverfahren zu konstruieren: Wende eine Saturierungsstrategie an und eliminiere gefährliche Fälle duch Vereinfachungsregeln. Um diesen Ansatz zu demonstrieren, präsentieren wir eine uniforme Beschreibung von mehreren wohlbekannten resolutionsbasierten Entscheidungsverfahren für das "Guarded Fragment", das zwei-Variablen und das monadische Fragmente ohne Gleichheit und zeigen, wie sie modular kombiniert werden können. Weiterhin beweisen wir, daß diese Verfahren die beste bekannte Komplexität besitzen.

Wir wenden unser Framework auf eine Reihe von Fragmenten der ersten Stufe, die mit Beschreibungslogiken verwandt sind, an. Zuerst betrachten wir eine einfache terminologische Sprache $\mathcal{EL}$, die lediglich Konjunktion und existentielle Restriktion erlaubt. $\mathcal{EL}$ ist effizient entscheidbar: Subsumption von Konzepten in $\mathcal{EL}$ ist bzgl. (zyklischer) Terminologien polynomiell entscheidbar [Baader, 2002] und behält diese

Eigenschaft unter vielen Erweiterungen bei [Baader et al., 2005]. Die polynomielle Entscheidbarkeit von $\mathcal{EL}$ kann teilweise dadurch erklärt werden, daß alle Klauseln, die durch Übersetzung von $\mathcal{EL}$-Terminologien entstanden sind, Hornklauseln sind. Dies ist allerdings nicht genug, da Hornlogik unentscheidbar ist. Die zweite wichtige Eigenschaft ist, daß die Skolemfunktionen, die in solchen Klauseln auftauchen, ihren nicht-funktionalen Teil eindeutig festlegen: Es ist nicht möglich, daß Klauseln $\neg A(x) \vee R(x, f(x))$ und $\neg B(x) \vee C(f(x))$ mit $A \neq B$ vorkommen. Es stellt sich heraus, daß dies eine Invariante unter Resolutionsinferenzen ist. Diese beiden Eigenschaften stellen sicher, daß geordnete Resolution mit angemessener Selektion lediglich kubisch viele Klauseln produziert. Das gibt uns auch Hinweise, wie man $\mathcal{EL}$ erweitert unter Erhaltung der polynomiellen Entscheidbarkeit.

Eine überraschende Eigenschaft von Saturierungsverfahren für $\mathcal{EL}$ ist, daß sie ohne Zuhilfenahme eines Theorembeweisers implementiert werden können. Dies ist möglich, da alle Resolutionsinferenzen für $\mathcal{EL}$ als Datalogregeln kodiert werden können, wie zum Beispiel:

$$\begin{array}{lccc} \text{Inferenz:} & \neg A(x) \vee \underline{B(f(x))};\ \neg \underline{B(x)} \vee C(x) & \vdash & \neg A(x) \vee C(f(x)) \\ \text{Wird kodiert durch:} & \mathsf{C}(A, B, f), \quad \mathsf{D}(B, C) & \rightarrow & \mathsf{C}(A, C, f). \end{array}$$

Um saturierungsbasierte Verfahren für ausdrucksstärkere Beschreibungslogiken zu erhalten, erweitern wir ein Entscheidungsverfahren von Ganzinger & de Nivelle [1999] für das Guarded Fragment. Das Guarded Fragment wurde von Andréka et al. [1996] eingeführt, um die guten Eigenschaften von modalartigen Sprachen auf Formeln der Logik erster Stufe zu übertragen. Man erhält das Guarded Fragment indem die Quantifizierungen der Formeln der Logik erster Stufe auf die folgende Form beschränkt werden: $\forall \overline{y}.[G \rightarrow F]$ oder $\exists \overline{y}.[G \wedge F]$, wobei G ein Atom-Guard ist, der alle freien Variablen von F enthält. Das Guarded Fragment enthält ausdrucksstärke Beschreibungslogiken, wie z.B. $\mathcal{ALCIH}$. Jedoch fallen einige Konstruktoren der Beschreibungslogiken, wie z.B. Nominale, funktionale Rollen und Zahlenrestriktionen, nicht in dem Guarded Fragment. Für jeden dieser Konstruktoren betrachten wir eine geeignete Erweiterung des Guarded Fragments.

Nominale entsprechen den Formeln der ersten Stufe der Form $\forall xy.[a(x) \wedge a(y) \rightarrow x \simeq y]$, die nicht Guarded sind. Nominale können alternativ durch die Formel $\forall x.[a(x) \rightarrow x \simeq c_a]$ ausgedrückt werden, was eine Guarded Formel ist, die eine Konstante c_a enthält. Um Nominale auszudrücken, ist es deshalb ausreichend, ein saturierungsbasiertes Verfahren auf das Guarded Fragment mit Konstanten zu erweitern. Dies kann auf zwei Arten geschehen: Durch Eliminierung von Konstanten, wie in [Grädel, 1999] beschrieben, oder direkt.

Bei der funktionalen Restriktion und den Zahlenrestriktionen gibt es das Problem, daß das Guarded Fragment mit Funktionalität im allgemeinen nicht entscheidbar ist [Grädel, 1999]. Da Rollen in Beschreibungslogiken größtenteils als Guards benutzt werden, betrachten wir eine Einschränkung des Guarded Fragments, bei der funktionale/Zahlen-beschränkungen nur auf die Guards angewandt werden. Es stellte sich heraus, daß diese Fragmente entscheidbar sind durch Paramodulation mit speziellen Vereinfachungsregeln.

Schließlich betrachten wir Erweiterungen des Guarded Fragments mit Kompositionsaxiomen der Form $S \circ T \subseteq H$. Es ist bekannt, daß solche Erweiterungen sehr problematisch sind, da Kompositionsaxiome in vielen Fällen unentscheidbar sind [Schmidt-Schauß, 1989; Grädel, 1999; Demri, 2001]. Gleichzeitig ist die Integration von Kompositionsaxiome eine Herausforderung, da viele Relationen durch solche Axiome charakterisiert werden: temporale Relationen, Ordnungen, Distanzen und topologische Relationen. Wir analysieren eine Vielzahl von Erweiterungen des Guarded Fragments mit Kompositionsaxiomen und wir identifizieren die Trennlinie zwischen "gefährlicher" und "sicherer" Anwendung solcher Axiome. Für unsere Entscheidbarkeitsbeweise benutzen wir eine erweiterte Variante des geordneten Verkettungskalküls von [Bachmair & Ganzinger, 1998*b*, 1995] und eine neue Art von Vereinfachungsregeln wie die folgende für Transitivität:

$$\texttt{TC}: \frac{\neg(xTy) \vee \neg a(x) \vee b(y)}{\begin{array}{r}\neg(xTy) \vee \neg a(x) \vee a^T(y) \\ \neg(xTy) \vee \neg a^T(x) \vee a^T(y) \\ \neg a^T(y) \vee b(y)\end{array}}$$

Im Gegensatz zu herkömmlichen Vereinfachungsregeln machen solche Regeln nicht Klauseln sondern Inferenzen redundant: Obige Regel führt ein neues Atom $a^T(x)$ ein, so daß (1) $a^T(y)$ in einem Schritt von $a(x)$ T-erreichbar ist, (2) $a^T(y)$ ist abgeschlossen unter T und (3) $a^T(y)$ ist enthalten in $b(y)$. Nach der Anwendung dieser Regel können wir "vergessen", daß T transitiv in den beteiligten Klauseln ist, was formal bedeutet, daß die Verkettungsinferenzen mit diesen Klauseln redundant werden.

Acknowledgements

First and foremost, this thesis would not be as it is without Prof. Dr. Harald Ganzinger, who accepted me as a PhD student into his group at the Max-Planck-Institute für Informatik. He was for me an example of a wise and rational researcher; he taught me automated deduction and gave me inspiration through his works, which have influenced the subject of this thesis. It is sad that he cannot appreciate what came out as the result of his ideas.

I am indebted to my supervisor Dr. Hans de Nivelle for his constant help and support in all my troubles, for patience with which he listened all my "inventions" and for his efforts in improving my scientific writing. Hans always had some time for me even during his deadlines. Several ideas of Hans have inspired and influenced some results of this thesis.

I thank Prof. Dr. Franz Baader for agreeing to be an external referee of my thesis and for interest to my works. I thank Prof. Dr. Gert Smolka for taking over me after Prof. Dr. Harald Ganzinger has passed away.

I am grateful to Prof. Dr. Andrei Voronkov who brought me into the area of automated reasoning and to Dr. Uwe Waldmann who taught me the "ins and outs" of saturation-based theorem proving. In particular, Uwe has patiently explained me all peculiarities for the notion of redundancy which, I hope I finally understood thanks to his help. I thank Dr. Viorica Sofronie-Stokkermans for reading the drafts of my papers and giving valuable remarks. Viorica has drawn my attention to DL $\mathcal{EL}$ which has eventually resulted in chapter 2 of this thesis. I thank Dr. Peter Baumgartner for encouragement, fruitful discussions and proof reading chapter 2 of this thesis. Peter and Dr. Patrick Maier have suggested to use XSB-prolog for implementation of datalog programs derived in chapter 2. Several people have encouraged my interests in description logics, among them were Prof. Dr. Franz Baader, Sebastian Brandt, Jan Hladik, Dr. Ulrike Sattler, and Anni-Yasmin Turhan. I am grateful to Birte Glimm for proof reading of the final version of this thesis.

This thesis has also profited from discussions on various topics with Thomas Hillenbrand, Dr. Konstantin Korovin, Dr. Chin Soon Lee, Dr. Renate Schmidt, Dr. Gernot Stenz, Alexander Malkis, Prof. Dr. Andreas Podelski, Dr. Stefan Ratschan and

Dr. Andrey Rybalchenko. I also thank Rostislav Rusev for helping me with running the experiments described in section 2.6. Special thanks to Brigitta Hansen, Kerstin Meyer-Ross, Simone Schulze, Veronika Weinand and administration of MPII for making my stay in Germany less troublesome as it could be.

A number of my friends have implicitly contributed to this thesis by encouraging/teasing me during all these years. Among them were Christoph Clodo, Kirill Dmitriev, Konstantin Korovin, Evghenia Stegantova, Andrey Rybalchenko, Dalibor Topic and my old Russian friends. Probably I have forgotten to mention somebody. Apologies to those.

And of course this thesis would not be possible without the permanent and warm support of my family.

Thank you all.

Biographical Sketch

Yevgeny Leonidovich Kazakov (Евгений Леонидович Казаков) was born in the city of Chelyabinsk, Russia on April 13th, 1977. Since his early childhood he showed interest in mathematics, which resulted in that he was accepted to a specialised High School for Physics & Mathematics (Lyceum No. 31), where he studied from 1990 to 1994. During this period he participated in many competitions on mathematics and physics and won several awards (most notably, the 1st place in the central Russian mathematical Olympiad).

In 1994 he was accepted to the Moscow State University on the Faculty of Mechanics and Mathematics where he took a series of courses on algebra, calculus, functional analysis, math statistics, optimisation theory and discreet mathematics. In 1996 he specialised in mathematical logic and worked on "logics of proofs" in the Department of Mathematical Logic and Theory of Algorithms under the supervision of Prof. Sergei N. Artemov and Prof. Vladimir N. Krupski. In June 1999 he graduated with honours. The title of his diploma thesis was: "Logics of Proofs for S5".

From 1999 to 2001 he worked in a company specialised on development and distribution of software solutions for accounting as a chief project manager. He obtained several specialist certificates on Russian software products for accounting.

In October 2001 he returned to research by joining the Programming Logics Group of the Max Planck Institute for Informatics in Saarbücken, Germany within the PhD programme of the International Max Planck Research School for Computer Science (IMPRS). He shifted his interests to automated deduction and later focused his research to saturation-based decision procedures and description logics. He is an author of several publications on these topics which include a solution of some open problem from description logics.

The following list gives in chronological order the refereed conference publications:[1]

[1] an up-to-dated list of publications, technical reports and drafts with respective links can be found in http://www.mpi-sb.mpg.de/~ykazakov/publications.html

Yevgeny Kazakov and Hans de Nivelle. Subsumption of concepts in $\mathcal{FL}_0$ for (cyclic) terminologies with respect to descriptive semantics is PSPACE-complete. In Diego Calvanese, Giuseppe De Giacomo, and Enrico Franconi, editors, *Description Logics*, volume 81 of *CEUR Workshop Proceedings*, 2003.

Yevgeny Kazakov and Hans de Nivelle. A resolution decision procedure for the guarded fragment with transitive guards. In David A. Basin and Michaël Rusinowitch, editors, *IJCAR*, volume 3097 of *Lecture Notes in Computer Science*, pages 122–136. Springer, 2004. ISBN 3-540-22345-2.

Yevgeny Kazakov. A polynomial translation from the two-variable guarded fragment with number restrictions to the guarded fragment. In José Júlio Alferes and João Alexandre Leite, editors, *JELIA*, volume 3229 of *Lecture Notes in Computer Science*, pages 372–384. Springer, 2004. ISBN 3-540-23242-7.

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, daß ich die vorliegende Arbeit selbständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus anderen Quellen oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet.

Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder ähnlicher Form in einem Verfahren zur Erlangung eines akademischen Grades vorgelegt.

Saarbrücken, den 5. Oktober 2005

Contents

List of Tables

List of Figures

List of Systems

Chapter 1

Introduction

One of the major areas of present day research is concerned with the development of systems for automated processing of information. The demand for such systems grows constantly along with the number of accessible information resources, especially for such huge ones as the World Wide Web.

One branch of the research in this area led to the development of so-called *knowledge representation languages*. The purpose of such language is to represent information in a rigorous and formally definable manner, and to be able to reason about it in a systematic and logical way, preferably with a minimal human interaction. Often such languages can be seen as *fragments of first-order logic*.

In this thesis we revisit the basic reasoning problems for such languages and look at them from the viewpoint of first-oder logic. We demonstrate that for many of these reasoning problems, it is possible to come up with *decision procedures* based on the theory of *saturation-based theorem proving* used in automated deduction.

The main part of this thesis is devoted to the development of such decision procedures for different *fragments of first-oder logic* (mainly extensions of the *guarded fragment*) that capture respective knowledge-representation formalisms. We argue that our approach has several principal advantages over the conventional *tableau-based* methods commonly used for reasoning in knowledge representation languages, which makes it more suitable for the development of reasoning procedures for very expressive languages such as the *ontology language for the Semantic Web* OWL.

1.1 Description Logics

Description logics (short DL) [Baader, Calvanese, McGuinness, Nardi & Patel-Schneider, 2003] is a family of formalisms for representation of and reasoning about *knowledge*. The basic blocks in description logics are *concepts* and *roles*. *Concepts* are thought as *sets* of entries (or *unary relations*). *Roles* represent *binary relations* on entries.

For example we can consider *concepts* Human, Male, Father which intuitively represent the set of all humans, respectively, male beings (including animals), and fathers. One can use formulas $\mathsf{Human}(x)$, $\mathsf{Male}(x)$, $\mathsf{Father}(x)$ to express the basic statements like "*x is a human*", "*x is male*" or "*x is a father*". Similarly, one can consider a *role* $\mathsf{has\text{-}child}$, and use $\mathsf{has\text{-}child}(x, y)$ to express "*y is a child of x*".

Although for us these *concepts* and *roles* carry already a certain meaning, for a computer they are not more than just words. This is because humans implicitly assume certain relationships between concepts, like, for instance:

$$\text{“}x \textit{ is a } \mathsf{Father} \quad \textit{if and only if} \\ x \textit{ is } \mathsf{Male}\textit{, } x \textit{ is a } \mathsf{Human}\textit{, and } x \textit{ } \mathsf{has\text{-}a\text{-}child} \textit{ } y \textit{ who is a } \mathsf{Human}\text{”} \tag{1.1}$$

These relationships are formalized in *description logics* by means of so-called *constructors*. For example, most of description logics allow for *conjunction of concepts*: $\langle\mathsf{concept}\rangle \sqcap \langle\mathsf{concept}\rangle$. This constructor can be used for defining, say, a new concept:

$$\mathsf{Man} \quad \doteq \quad \mathsf{Human} \sqcap \mathsf{Male} \tag{1.2}$$

as the *intersection* for the sets of Human and Male beings. Binary relations participate in *role constructors*, using which, say, (1.1) can be formalized as follows:

$$\mathsf{Father} \quad \doteq \quad \mathsf{Human} \sqcap \mathsf{Male} \sqcap \exists \mathsf{has\text{-}child}.\mathsf{Human} \tag{1.3}$$

Here *role* $\mathsf{has\text{-}child}$ is used in a *constructor* $\exists\langle\mathsf{role}\rangle.\langle\mathsf{concept}\rangle$ called *existential restriction*. Concept $\exists\mathsf{has\text{-}child}.\mathsf{Human}$ represents the set of all elements x such that there *exists* some y related to x by *role* $\mathsf{has\text{-}child}$ such that $\mathsf{Human}(y)$ holds.

Different description logics typically use different sets of *constructors*. The more constructors there are in a description logic—the more *expressive* the logic is.

A collection of *definitions* of the above form is called a *terminology* and represents a basic *knowledge* about *concepts*. In order to process knowledge on a computer, it is not enough to provide constructors for writing terminologies; one should also be able to *reason* about concepts. For example, a computer must be able to conclude that the following simplified definition describes *the same* concept as (1.3):

$$\mathsf{Father} \quad \doteq \quad \mathsf{Man} \sqcap \exists \mathsf{has\text{-}child}.\mathsf{Human} \tag{1.4}$$

Such *reasoning* is possible using dedicated procedures that are designed for description logics of interest. These algorithms typically address the following questions about a *terminology*: *"are concepts C and D equivalent?"*, or *"is every element of*

C belongs to D?". Such questions are harder to answer for *expressive* description logics.

Originally ontologies have been developed in order to provide *meanings* to information entries. In description logics, this can be done using so-called *assertions* which annotate entries with relevant concepts. For example, the assertion:

$$\mathit{Bill} \ : \ \mathsf{Father}$$

describes *individual Bill* as an *instance* of concept Father. This assignment has implicit consequences, for example that *Bill* must be Male, that has-child, etc.

1.2 Saturation-Based Decision Procedures

Historically, the research in expressive description logics went into two separate directions. One line of research has been concerned with *complexity* issues for description logics [see Donini, 2003; Tobies, 2001]. The other line has been focused on the development of *practical procedures* for solving reasoning tasks in description logics [see e.g., Horrocks et al., 2000]. Such procedures typically exhibit suboptimal worst-case complexity, however they were proven to behave good in practice because of numerous *optimisation techniques* employed.

The difference between these two approaches is that they are often based on different reasoning paradigms: *practical procedures* typically employ *tableau-based procedures* that (attempt to) construct a model; *optimal* algorithms often apply automata-theoretic arguments to extract complexity results. Although there have been some attempts to bring these two approaches under the same roof [see e.g., Baader, Hladik, Lutz & Wolter, 2003], so far there is no practical reasoner for *expressive* description logics, which has a provably optimal complexity. In this thesis we argue that *saturation-based theorem proving* could serve this unifying rôle by providing *theoretically optimal* procedures that are *amenable to optimisation*.

What are the saturation-based procedures? Procedures based on saturation are *general* methods for checking *satisfiability/validity* of *first-order formulas* [Bachmair & Ganzinger, 2001]. These procedures have been implemented in many *automated theorem provers* like SPASS [Weidenbach, Brahm, Hillenbrand, Keen, Theobalt & Topić, 2002] and VAMPIRE [Riazanov & Voronkov, 2002]. Saturation procedures work by producing inferences from formulas using special *inference rules* like:

Resolution

$$\mathtt{R}: \frac{C \vee A \qquad D \vee \neg A}{C \vee D}$$

This process of producing new formulas from existing ones is called *saturation.* There are three possible outcomes of the saturation procedure: either (**1**) a "*contradiction*" is obtained or (**2**) a set of formulas (the *saturation*) is computed which is closed under all rules, or (**3**) the procedure does not terminate. The first two outcomes correspond to "Yes/No" answers for the input problem (is formula *satisfiable/valid* ?).

How do description logics relate to first-order logic? The major advantage of description logics over the previous languages for knowledge representation, is the presence of well-defined *formal semantics* [Levesque & Brachman, 1987]. This means, in particular, that there is a native correspondence between *constructors* of description logics and *first-order formulas*, for example: $\mathsf{Human} \sqcap \mathsf{Male} \dashrightarrow \mathsf{Human}(x) \wedge \mathsf{Male}(x)$, $\exists \mathsf{has\text{-}child}.\mathsf{Human} \dashrightarrow \exists y.[\mathsf{has\text{-}child}(x, y) \wedge \mathsf{Human}(y)]$, etc. This translation opens possibilities of using general first-order theorem provers for reasoning in description logics: one simply applies this translation, and runs a prover on the resulted formulas.

Unfortunately, a general prover does not terminate in general. Yet, for formulas of certain *restricted forms* (say, for formulas that correspond to a particular description logic) it is often possible to insure termination by "tweaking" certain parameters of a prover. In other words, a prover can be turned into a *decision procedure* for a specific *class* of formulas, and hence, can be used for solving reasoning tasks in particular description logics.

What is the advantage of saturation-based procedures? All procedures need to be shown correct (at least once). In order to prove correctness, one generally needs to show three things: (**1**) *soundness*, (**2**) *completeness* and (**3**) *termination* of the procedure. The first two mean that whenever a "Yes/No" answer is computed by a procedure then this is the right answer.

In order to prove *soundness* and *completeness* for *tableaux algorithms*, one often needs to deal with a rather non-trivial analysis of models. Such analysis is hard to formalise, and, what is more important, all proofs must be reconsidered every time the procedure is changed (say, in order to cope with new constructors).

In contrast to tableaux-based procedures, soundness and completeness of a general saturation procedure is proven *once-and-for-all.* Every specific strategy for a particular class of formulas is an *instance* of the *general* procedure, which means that it is automatically *sound* and *complete.* Hence proving correctness of saturation-based procedures is essentially reduced to proving (3) *termination.*

1.3 The Guarded Fragment and Its Extensions

As has been pointed out, description and modal logics correspond to *fragments* of first-order logics via their standard semantical translation. Andréka et al. [1996] have noticed that most formulas from such fragments are *guarded*.

The guarded formulas are constructed as usual first-order formulas, with one exception: all quantification in such formulas must be *bounded*, i.e., of the form:

$$\forall\overline{y}.[G \rightarrow F] \qquad \textit{or} \qquad \exists\overline{y}.[G \wedge F]$$

where G is an atom-*guard* for F, meaning that G contains *all free variables* of F. For example the formula $\exists y.[\mathsf{has\text{-}child}(x,y) \wedge \mathsf{Human}(y)]$ that corresponds to concept $\exists\mathsf{has\text{-}child}.\mathsf{Human}$, has a *guard* $\mathsf{has\text{-}child}(x,y)$. The set of all *guarded formulas* forms the *guarded fragment*.

The *guarded fragment* has been extensively studied in literature. Grädel [1999] has demonstrated that this fragment has many nice model-theoretic properties, which make it more "close" to modal and description logics than the *two-variable* fragment, considered previously. He has also described a *decision procedure* for the *guarded fragment* and characterised its complexity. Ganzinger & de Nivelle [1999] have found an elegant saturation-based decision procedure for the *guarded fragment*, which opened possibilities for the *practical* usage of this fragment. In this thesis we generalise and extend this procedure in various ways.

Although many description logics can be expressed in the *guarded fragment*, there are some constructors which do not correspond directly to guarded formulas, like *nominals*, *functional restrictions for roles* and *(qualified) number restrictions*. In this thesis we discuss how to *extend* the guarded fragment in order to cope with these constructors, and how to obtain saturation-based decision procedures for these extensions.

1.4 Theories of Compositional Binary Relations

One particular type of constructors in description logics which is especially hard to deal with, is characterised by *compositional axioms* of form:

$$S \circ T \subseteq H \qquad \dashrightarrow \qquad \forall xyz.[(xSy) \wedge (yTz) \rightarrow (xHz)] \tag{1.5}$$

Such axioms are often required when we want to speak about relations with *complex dependencies*, such as *temporal* relations, *orderings*, *geometrical* or *topological* relations. For example, the following properties can be seen as instances of (1.5):

- If (x happened before y) and (y happened before z)
  then (x happened before z)

- If $(x < y)$ and $(y \leq z)$ then $(x < z)$
- If (x is a $\mathsf{part\text{-}of}$ y) and (y is $\mathsf{located\text{-}in}$ z) then (x is $\mathsf{located\text{-}in}$ z)
- If (the $\mathsf{distance}$ between x and y is ≥ 5) and
  (the $\mathsf{distance}$ between y and z is < 2)
  then (the $\mathsf{distance}$ between x and z is ≥ 3)

Integration of such *compositional theories* into *description logics* is often required for medical applications [Schulz & Hahn, 2001; Rector, 2002], and would open new perspectives for terminological reasoning about *ordered domains*, *temporal processes*, *and geographical data*.

The difficulty is that compositional axioms often lead to *undecidability* of logical formalisms. Already the *transitivity axiom* $T \circ T \subseteq T$ when added to the *guarded fragment*, makes it undecidable [Grädel, 1999]. Many simple description logics loose decidability with *general* compositional axioms of form (1.5) [Schmidt-Schauß, 1989; Baader, 2003; Donini, 2003].

In this thesis we study several extensions of the guarded fragment with compositional theories and identify the *decidability barrier* for these extensions. The crucial observation which allows us to establish many decidability results, is that compositional theories from the "real world" are not "general", but admit many *regular algebraic properties* (for example *associativity* of composition), which can be exploited in decision procedures.

1.5 Outline and Structure of this Thesis

This thesis is devoted to application of the Bachmair & Ganzinger's [1990; 1994] framework of saturation-based theorem proving to the decision problem for extensions of the *guarded fragment* that are relevant for *description logics* and other knowledge representation languages. The main points of this thesis can be put as follows:

- We give a *uniform* account of many known and new saturation-based decision procedures. We introduce a special *scheme notation* which facilitates description of such procedures. We also show that our procedures give the best known *complexities* in most cases.
- We argue that many fragment-specific strategies used in saturation-based procedures, can be naturally formulated as *simplification rules*. Simplification rules have an advantage that they do not affect refutational completeness of

saturation-based calculi and may be combined with each other in arbitrary ways.

- We introduce a new form of *combination* of first-order fragments which naturally corresponds to combining of constructors in description logics. We argue that such combinational approach makes it possible to design decision procedures in a modular way.

- Finally, we introduce a new class of simplification rules, which unlike standard simplification rules affect *inferences* and not *clauses*. We use such simplification rules for obtaining decision procedures for several extensions of the guarded fragment with *compositional theories*.

The main material of this thesis is organized as follows:

In chapter 2 we give a preview of our methods on the example of a small terminological language $\mathcal{EL}$. Despite its introductory status, this chapter describes several *new* (un)decidability and complexity results for extensions of $\mathcal{EL}$, and provides an empirical comparison of saturation and tableaux-based procedures. This gives a strong motivation for our subsequent theoretical studies.

In chapter 3 we give standard technical preliminaries to the material of this thesis. We introduce logical terminology, non-classical logics, decidable fragments of first-order logic and domino problems. Then we introduce the framework of refutational theorem proving which is the theoretical basis of our decision procedures.

In chapter 4 we describe several saturation-based decision procedures for the guarded, two-variable and monadic fragments, their combinations and extensions.

First, we are concerned with *resolution-based decision procedures*: we refine the known procedures for the *guarded*, *two-variable* and *monadic* fragments without equality and demonstrate that our procedures have the *best known complexity*.

Then we introduce a new type of *combination* of fragments which composes their recursive definitions, and argue that such combination is useful for *modular* development of decision procedures: we demonstrate how resolution decision procedures for combinations of the *guarded*, *two-variable* and *monadic* fragments can be obtained by reusing the procedures for their components.

After that, we focus on extensions of the guarded fragment *with equality*. We consider several extensions that correspond to constructors frequently used in *description loigc*: the guarded fragment with *constants*, *functionality* and *number restrictions*, and extend the procedure of Ganzinger & de Nivelle [1999] for the guarded fragment with equality, to these fragments.

In chapter 5 we focus on extensions of the guarded fragment with *compositional theories* characterised by axioms of form $S \circ T \subseteq H$. This chapter presents case studies of (un)decidability/complexity results for many of these extensions.

First we consider extensions of the guarded fragment without equality. From [Grädel, 1999; Ganzinger, Meyer & Veanes, 1999] it is known that the guarded fragment with transitivity is in general *undecidable*. We sharpen both of these undecidability results by proving that already *two* transitive relations make the *two-variable* version of the guarded fragment undecidable.

Following Szwast & Tendera [2001] we consider a restricted version of the guarded fragment with transitivity—the *guarded fragment with transitive guards*—in which transitive relations are restricted to occur only as guards. Inspired by a translation of modal formulas over transitive frames proposed in [de Nivelle, 1999], we formulate a special simplification rule which makes certain dangerous inference with transitive atoms redundant. This additional rule makes it possible to obtain a decision procedure for the *guarded fragment with transitive guards* based on the *chaining calculus*.

After that, we generalise our procedure to a larger class of compositional axioms which admit *associativity property* and relax restrictions on usage of compositional relations in guarded formulas. We show then that further possible extensions of the guarded fragment with "*regular*" compositional axioms of form $S \circ T \subseteq H_1 \cup \cdots \cup H_n$ are not decidable in general.

Finally we reconsider our results for the case when equality is allowed, and demonstrate that most decidability results are no longer valid. The only exception is the *guarded fragment with transitive guards and equality* [Szwast & Tendera, 2001], for which we sketch a saturation-based procedure based on special simplification rules.

In chapter 6 we summarise all results obtained in this thesis and outline possible direction for future works.

Some technical material from chapters 2, 4 and 5 is postponed to Appendixes A, B, C and D. This thesis tries to be self-contained, however the necessary references to the related literature are also provided to complement the material of the thesis. Some results mentioned in this thesis have been previously published in [Kazakov & de Nivelle, 2003, 2004; Kazakov, 2004].

1.6 Contributions

The following original results have been presented in this thesis:

1. Polynomial-time *saturation-based* procedures for solving subsumption and instance problems in extensions of DL $\mathcal{EL}$ with GCIs, *role hierarchies*, *conjunctions of roles*, *cross-products of concepts*, *nominals* and *restricted role-value maps*.[1] Empirical evaluation of the procedures for $\mathcal{EL}$ on randomly generated TBoxes. [chapter 2]

2. Undecidability for DL $\mathcal{EL}$ with *restricted role-value maps* extended with one of the following constructors: *conjunction of roles*, *disjunction of concepts*, *universal value restriction*, and *inverse roles*. [chapter 2]

3. Resolution-based decision procedures for the *guarded*, *two-variable* and *full monadic* fragments without equality, of best known complexities.[2] [chapter 4]

4. The notion of *structural combination* of recursively defined fragments of first-oder logic. Resolution-based decision procedures for combinations of the *guarded*, *two-variable* and *full monadic* fragments without equality. Undecidability for some combinations of these fragments in the case with equality. [chapter 4]

5. Paramodulation-based decision procedures for extensions of the *guarded fragment with equality*, *constants* and *counting guards*. A related result has been published in [Kazakov, 2004]. [chapter 4]

6. Undecidability for the *two-variable guarded fragment* without equality with two transitive relations. [chapter 5]

7. Complexity-optimal chaining-based decision procedures for the *guarded fragment* without equality with *conjunctions of compositional guards*. A similar resolution-based procedure that covers a part of this result has been published in [Kazakov & de Nivelle, 2004]. [chapter 5]

8. Undecidability for: the *two-variable guarded fragment without equality*, with guards from a *relational algebra*; the *two-variable guarded fragment with equality* and *compositional guards*; and the *two-variable guarded fragment with equality* and *conjunctions of transitive guards*. [chapter 5]

[1] polynomial-time subsumption algorithms for some of these DLs were known from [Baader, 2003; Brandt, 2004*a*; Baader et al., 2005]

[2] decidability of these fragments by resolution was known from other works, however the complexities of such procedures were either not known or not optimal for some cases

Chapter 2

Engineering Logical Algorithms using Saturation-Based Theorem Proving

In this chapter we demonstrate how the framework of saturation-based theorem proving of [Bachmair & Ganzinger, 1994, 1990] can be applied for solving certain problems in the area of knowledge representation and reasoning (KR&R). We consider a simple *description logic* $\mathcal{EL}$ and show how standard reasoning problems for this description logic can be solved using the *ordered resolution calculus*. We also discuss how these procedures can be extended to more expressive languages. The goal of this chapter is to give some motivations and to make an overview of the techniques that will be used and justified throughout this thesis. Despite its introductory status, this chapter presents *new* results for description logics.

$\mathcal{EL}$ is a simple description logic that allows for concept construction using only *conjunctions* and *existential restrictions*. Despite its limited expressive power, $\mathcal{EL}$ can be practically used for conceptual modelling, in particular, over a widely used medical terminology SNOMED [Spackman, Campbell & Cote, 1997]. Baader [2002] made a surprising observation, that the *concept subsumption problem* for this description logic is only *polynomial* even w.r.t. *cyclic terminologies* – a quite unexpected result for such expressive languages as description logics. Even for a similar description logic $\mathcal{FL}_0$ that allows for *conjunctions* and *universal restrictions*, or its dual variant that allows for *disjunctions* and *existential restrictions*, reasoning is already PSPACE-hard [Baader, 1996; Kazakov & de Nivelle, 2003].

It is well-known that modal and description logics can be seen as fragments of first-order logic via the standard semantical translation. This makes it possible to apply first-order theorem provers for solving reasoning problems associated with these logics, in particular, satisfiability of modal formulas or subsumption of concepts in description logics. It has been observed quite long ago that theorem provers based on *ordered resolution*, a refinement of Robinson's [1965] resolution calculus, can be turned into a *decision procedure* for many fragments of first order logic [Joyner Jr., 1976; Fermüller, Leitsch, Tammet & Zamov, 1993] and first-order counterparts of modal logics [see e.g., Schmidt & Hustadt, 2003].

The question, we are going to answer in this chapter is, can these methods based on theorem provers explain tractability of $\mathcal{EL}$, and if so, which extensions this description logic might have that retain its low complexity? It is well-known that first-order translations often destroy the structure of modal formulas, which results in higher complexity for resulted decision procedures. Indeed, the basic modal logic **K** is PSPACE-complete, whereas most of the known resolution decision procedures for **K** require exponential space and time.[1] Fortunately, this does not happen with $\mathcal{EL}$. We were able to explain tractability of $\mathcal{EL}$ and many of its extensions found in [Baader, 2002, 2003; Brandt, 2004*a*; Baader et al., 2005]. Moreover, we have found *new* polynomial extensions of $\mathcal{EL}$.

Our method uses the ordered resolution calculus for obtaining decidability and complexity results. Surprisingly, it turns out that no resolution-based theorem prover is actually required to *implement* these decision procedures. We have found out that our procedures can be specified more concisely as *datalog programs*. As a consequence, $\mathcal{EL}$-knowledge bases can be turned into conventional (relational) databases, so that all standard reasoning tasks for description logic are reduced to query evaluation in *datalog*. This, on one hand, provides us "for free" with efficient implementations for these description logics, since any optimised "off-the-shelve" deductive database system (say, XSB-prolog [Sagonas, Swift & Warren, 1994]) can do the hard job. On the other hand, translation to datalog allows one to extract upper complexity bounds for the worst case running times for our procedures by known techniques from static analysis of logical programs [McAllester, 2002]. Importantly, our reduction to datalog is almost mechanical, which means that correctness of our decision procedures relies only on refutational completeness of the ordered resolution calculus, which is proven "once and for all".

2.1 Description Logic $\mathcal{EL}$

The syntax of the *description logic* (short DL) *with existential restrictions* $\mathcal{EL}$ consists of a set CN of *concept names* and a set RN of *role names* which are the primitive constructors. Intuitively, every concept name $A \in \mathrm{CN}$ stands for a set and every role name $R \in \mathrm{RN}$ stands for a binary relation. Description logic $\mathcal{EL}$ reasons about compound *concepts* (= sets) that are constructed from these base elements as follows:

[1] Except for hyper-resolution strategies with splitting and backtracking that essentially simulate tableau procedures for **K** [see Hustadt, 1999; Schmidt & Hustadt, 2003]

- *top concept* $\top$ is a concept, which intuitively represents the set of all elements;
- every concept name $A \in \mathrm{CN}$ is a concept;
- if C_1 and C_2 are concept then so is $(C_1 \sqcap C_2)$, which intuitively represents the intersection of the sets C_1 and C_2;
- if C_1 is a concept and $R \in \mathrm{RN}$ is a role name then $\exists R.C_1$ is a concept for the set of all elements that are related to some element from C_1 via R.

In abstract syntax, we often write this definition as follows:

$$\mathcal{EL} ::= \top \mid A \mid C_1 \sqcap C_2 \mid \exists R.C_1 \ . \tag{2.1}$$

where $A \in \mathrm{CN}$, $R \in \mathrm{RN}$ and $C_1, C_2 \in \mathcal{EL}$.

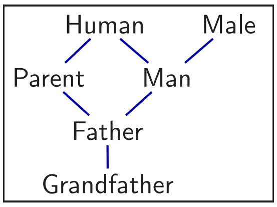
The language of description logics is used to formulate logical descriptions for deferent notions by their mutual relationship. For example, we can *define* a concept Man by writing $\mathsf{Man} \doteq \mathsf{Human} \sqcap \mathsf{Male}$, which means that every Man is a Male and Human, and vice versa. Note that this is not a definition in the usual sense, since concepts Human and Male are not (yet) defined and might be not defined at all. To continue our example, suppose we additionally have a role name has-child which we think of, as describing the parent–child relation: $(x, y) \in \mathsf{has\text{-}child}$ *iff* "y is a child of x". Using this role name, we can define a concept Parent as those Humans that have a Human offspring: see Table 2.1. Similarly, we define a concept Father as a Man that has a Human child, and a concept Grandfather as a Man that has a child who is a Parent. In general, a concept definition has a form $A \doteq C$ where $A \in \mathrm{CN}$ is a concept name and $C \in \mathcal{EL}$ is a concept constructed according to (2.1).

Table 2.1 An $\mathcal{EL}$-terminology of human relations

TBox	ABox
$\mathsf{Man} \doteq \mathsf{Human} \sqcap \mathsf{Male}$ $\mathsf{Parent} \doteq \mathsf{Human} \sqcap \exists\mathsf{has\text{-}child}.\mathsf{Human}$ $\mathsf{Father} \doteq \mathsf{Man} \sqcap \exists\mathsf{has\text{-}child}.\mathsf{Human}$ $\mathsf{Grandfather} \doteq \mathsf{Man} \sqcap \exists\mathsf{has\text{-}child}.\mathsf{Parent}$	$\mathit{John} : \mathsf{Man}$ $\mathit{Bill} : \mathsf{Father}$ $(\mathit{John}, \mathit{Bill}) : \mathsf{has\text{-}child}$

A collection of concept definitions forms a *terminology*, or TBox (abbreviated from *terminological box*) and represents the general knowledge about different notions = concepts. In addition, one can provide more specific information by specifying instances of concepts, which are called *individuals*, and role relations between them. For example, we can specify an individual *John* as an instance of concept Man, and an individual *Bill* as a Father and a child of *John* (see the right column of Table 2.1). Assertions can be of two forms: either (***i***) $a : C$ called a *concept assertion*, where a is an individual and C is a concept, or (***ii***) $(a, b) : R$ called a *role assertion*, where a and b are individuals and R is a role name. A collection of *assertions* of these forms is called an *assertion box*, or short ABox.

Apart from the information stated in definitions, terminologies and assertions *imply* implicit relationships between individuals and concepts. It is logically clear from the definitions in Table 2.1 that every Father must be a Parent since every Man is a Human according to the first definition. In this case, we say that concept Parent *subsumes* concept Father (or Father *is subsumed by* Parent) and write Father $\sqsubseteq$ Parent. Intuitively, subsumption of concepts means inclusion for the sets of elements, that correspond to these concepts. Returning to our example, one can show that concept Grandfather is subsumed by concept Father (in other words, every Grandfather is a Father), since every Parent is a Human according to the second definition from Table 2.1, and hence, every Man that has a Parent child, has automatically a Human child. Now looking at ABox, we can observe that *Bill* is a Parent, since we have proved that Father $\sqsubseteq$ Parent. So *John* must be a Grandfather because he is a Man and has a child *Bill* who is a Parent.



Description logic systems (DL-systems) are specially designed to carry out such reasoning about implicit information present in knowledge bases. The main reasoning task for terminologies is *classification of concepts*, whose purpose is to arrange the concepts in a hierarchical structure, called a *taxonomy*, according to subsumption relation. The graph to the right represents a taxonomy of concepts defined in Table 2.1, where each concept subsumes all connected concepts that are situated lower in the graph. Classification of concepts is useful for optimisation of *queries* about *individuals*. For example, our terminology can be queried by asking "?- x : Human" for all instances of concept Human. The answer to this query is $\{John, Bill\}$, which can be easily obtained using the computed taxonomy. Apart from classification, description logic systems provide for other *reasoning services*, the most important of which are listed in Table 2.2.

For the particular description logic $\mathcal{EL}$, the *concept satisfiability* and *concept disjointness* problems are trivial, i.e., the answers are always "*Yes*" and "*No*" respectively: basically, it is always possible that all concepts consist of all elements and every role is a total relation. However, for certain extensions of $\mathcal{EL}$, that we consider, these reasoning problems make sense. *Concept equivalence* can be reduced to *concept subsumption* by an easy observation that $C \equiv D$ *iff* $C \sqsubseteq D$ and $D \sqsubseteq C$. Taxonomy of concepts can be also reconstructed using concept subsumption queries by (roughly) checking subsumption relation between every pair of concepts. Hence,

Table 2.2 Some reasoning problems for description logics

TBox-reasoning:	
Concept satisfiability:	*Given a concept C, check whether C may contain some element:* ?- $A = \bot$;
Concept subsumption:	*Given two concepts C and D, check whether C is always a subset of D:* ?- $C \sqsubseteq D$;
Concept equivalence:	*Given two concepts C and D check whether they stand for the same set:* ?- $C \equiv D$;
Concept disjointness:	*Given two concepts C and D check if they are disjoint:* ?- $C \sqcap D \equiv \bot$;
ABox-reasoning:	
Instance problem:	*Given an individual a and a concept C check if a is an instance of C:* ?- $a : C$;
	Given two individuals a and b and a role name R, check if a and b are connected with R: ?- $(a, b) : R$;
Retrieval problem:	*Given a concept C find all individuals a that are instances of C:* ?- $x : C$.

concept subsumption is a central reasoning problem in description logics.

Baader [2002] came up with an algorithm for checking subsumption between concepts using a certain graph-theoretic characterisation. He showed that subsumption between two concepts is equivalent to existence of a simulation relation between so-called *description graphs* that correspond to terminological axioms in a *normal form*. The last can be computed in polynomial time in the size of graphs, yielding, a polynomial subsumption algorithm. Subsumption algorithms based on such kind of characterisation are usually called *structural subsumption algorithms*, because they compare the *syntactical structure* of definitions.[2] Such subsumption algorithms have been employed in the first generation of DL-systems for families of Kl-One knowledge representation languages [Brachman & Schmolze, 1985]. Although the structural subsumption algorithms were quite efficient, they were complete only for very restrictive languages.

Starting from the work of Schmidt-Schauß & Smolka [1991], the majority of modern DL-systems employed so-called *tableau-based procedures*. These procedures attempt to construct a model step-by-step that satisfies all terminological axioms and assertions. Tableau procedures have been extended to a variety of expressive description logics and found to be amenable to optimisations [see Horrocks et al., 2000]. Nowadays there are several efficient implementations of tableau procedures for very expressive description logics available, notably, Fact [Horrocks, 1998] and Racer [Haarslev & Möller, 2001].

[2] See [Baader & Nutt, 2003] for an overview and comparison of different algorithms for reasoning in description logics

2.2 Resolution-Based Decision Procedures

Tableau-based decision procedures for expressive description logics have become rather involved: they do no longer construct a model explicitly, but search for a model in some *representation*. Because of this, correctness of the procedures is hard to justify formally, especially for those description logics with many constructors. There are some well-known examples where a subtle interaction between different constructors[3], or not appropriate understanding of their semantics[4] cause incorrectness of straightforward tableau algorithms. To avoid possible pitfalls of that kind, a formal analysis of these algorithms is required. However, there is an obvious lack of formal tools to carry out such analysis, which, although, not crucial for simple description logics, might become a serious problem for more expressive languages with many constructors, like $\mathcal{SHIQ}$, OWL DL and their extensions.

In this thesis we advocate another approach for engineering of reasoning procedures in DLs, that might help to overcome these difficulties, namely by using *saturation-based theorem proving*. The advantage of saturation-based procedures over tableau-based algorithms for description logics is that (**1**) they utilise the semantics of DL-constructors *directly*, and (**2**) employ general sound and complete calculi.

The idea of using first-order theorem provers, in particular those based on *ordered resolution* for deciding modal and description logics, is not new and there have been quite a large body of works on this topic (see e.g., [Schmidt & Hustadt, 2003] for an overview). However, to the best of our knowledge, there are only few implementations of systems based on resolution and only for limited classes of modal and description logics (notably MSpass [Hustadt, Schmidt & Weidenbach, 1999]). These systems are currently not efficient enough and cannot cope with expressive languages. In this thesis we present theoretical and imperial evidences that this picture might change.

A distinguished feature of description logics from previous generations of knowledge representation languages (like *semantic networks* and *frames*), is the presence of well-defined *formal semantics* for all constructors. Semantics formulates the precise *meaning* for concept descriptions and reasoning problems. For the purpose of our presentations, we do not define semantics of $\mathcal{EL}$, but give a first-order counterparts for all of its elements. Then semantics of $\mathcal{EL}$ is a standard semantics for first-order logic.

A correspondence between description logic concepts and first-order formulas, is defined by associating a *unary atom* $A(x)$ with every concept name $A \in \mathrm{CN}$, and a

[3] See, for example, [Horrocks et al., 2000] demonstrating how the usage of *non-simple* roles in number restrictions leads to undecidability of subsumption in $\mathcal{SHIQ}$

[4] See examples and related discussion in [Tobies, 2001, Chapter 4]

binary atom $R(x, y)$ with every role name $R \in \text{RN}$. This mapping is extended over definition (2.1) to arbitrary $\mathcal{EL}$-concept as follows:

$$\mathcal{FO}(\mathcal{EL}) ::= \top \mid A(x) \mid C_1(x) \wedge C_2(x) \mid \exists y.[R(x, y) \wedge C_1(y)] \; . \tag{2.2}$$

where $A \in \text{CN}$, $R \in \text{RN}$ and $C_1(x), C_2(x) \in \mathcal{FO}(\mathcal{EL})$.

For example, a concept in the right hand side of the definition for Parent from Table 2.1 is translated to formula $\mathsf{Human}(x) \wedge \exists y.[\mathsf{has\text{-}child}(x, y) \wedge \mathsf{Human}(y)]$. As seen from (2.2), every $\mathcal{EL}$ concept corresponds to a first-order formula with one *free variable*. The translation maps a TBox-definition $A \doteq C$ to an axiom $\forall x.(A(x) \leftrightarrow C(x))$. Every ABox-assertion $a : C$ and $(a, b) : R$, yields axiom $C(a)$ and $R(a, b)$ respectively, where individuals a and b are treated as constants. Consequently, TBox and ABox correspond to conjunction of first-order formulas of the above forms.

To simplify the upcoming exposition of our procedures for $\mathcal{EL}$, we assume that every definition in a TBox and every assertion from an ABox is *simple*, i.e., for every definition $A \doteq C_1 \sqcap C_2$ or $A \doteq \exists R.C_1$, concepts C_1 and C_2 are *concept names*, and for every concept assertion $a : C$, concept C is a *concept name* either. This is not a severe restriction, since every TBox can be translated in linear time into an equivalent TBox with only simple definitions, by introducing new concept names for every compound concept. For example, definitions from terminology given in Table 2.1 can be pre-processed as follows:

$$\begin{array}{rclrcl} \mathsf{Man} & \doteq & \mathsf{Human} \sqcap \mathsf{Male} & \mathsf{Father} & \doteq & \mathsf{Man} \sqcap \mathsf{A} \\ \mathsf{A} & \doteq & \exists\mathsf{has\text{-}child}.\mathsf{Human} & \mathsf{B} & \doteq & \exists\mathsf{has\text{-}child}.\mathsf{Parent} \\ \mathsf{Parent} & \doteq & \mathsf{Human} \sqcap \mathsf{A} & \mathsf{Grandfather} & \doteq & \mathsf{Man} \sqcap \mathsf{B} \end{array} \tag{2.3}$$

In Table 2.3 (please ignore the last two columns for now), we have summarised the first-order translation for simplified $\mathcal{EL}$-terminologies and assertions (definitions of form $A \doteq C$ are split into two *concept inclusion axioms*: $A \sqsubseteq C$ and $A \sqsupseteq C$).

2.2.1 The Ordered Resolution Calculus

A resolution-based theorem prover does not work with first-order formulas directly, but with *clauses*. A clause is an expression of form $C = L_1 \vee \cdots \vee L_n$, where each *literal* L_i with $1 \leq i \leq n$ is either an atom $A(x_1,.., x_k)$ or a negation of an atom $\neg A(x_1,.., x_k)$. Intuitively, a clause corresponds to a first-order formula $\forall x_1 \cdots x_k.[L_1 \vee \cdots \vee L_n]$, where $\{x_1,.., x_k\}$ are all variables that occur in the clause. In other words, all variables in clauses are *implicitly universally quantified*. A *set of clauses* is understood as conjunction of the formulas that correspond to clauses.

Transformation of a formula into a *clause normal form* (short **CNF**), is a routine consisting of propositional simplifications and elimination of quantifiers from the

Table 2.3 A first-order translation and **CNF**-translation for $\mathcal{EL}$-terminologies

TBox-*definition*	$\mathcal{FO}$ – *translation*	**CNF** – *translation*	*clause type*
$A \doteq \top$	$\forall x.[A(x) \leftrightarrow \top]$	$A(x)$	$\mathsf{C3}(A)$
$A \sqsubseteq (B \sqcap C)$	$\forall x.[A(x) \rightarrow (B(x) \wedge C(x))]$	$\neg A(x) \vee B(x)$ $\neg A(x) \vee C(x)$	$\mathsf{D4}(A, B)$ $\mathsf{D4}(A, C)$
$A \sqsupseteq (B \sqcap C)$	$\forall x.[(B(x) \wedge C(x)) \rightarrow A(x)]$	$\neg B(x) \vee \neg C(x) \vee A(x)$	$\mathsf{D5}(B, C, A)$
$A \sqsubseteq \exists R.B$	$\forall x.[A(x) \rightarrow \exists y.(R(x, y) \wedge B(y))]$	$\neg A(x) \vee R(x, f_A(x))$ $\neg A(x) \vee B(f_A(x))$	$\mathsf{C5}(A, R, f_A)$ $\mathsf{C4}(A, B, f_A)$
$A \sqsupseteq \exists R.B$	$\forall x.[\exists y.(R(x, y) \wedge B(y)) \rightarrow A(x)]$	$\neg R(x, y) \vee \neg B(y) \vee A(x)$	$\mathsf{D6}(R, B, A)$
ABox-*assertion*	$\mathcal{FO}$ – *translation*	**CNF** – *translation*	*clause type*
$a : A$ $(a, b) : R$	$A(a)$ $R(a, b)$	$A(a)$ $R(a, b)$	$\mathsf{C1}(A, a)$ $\mathsf{C2}(R, a, b)$

formula. Universal quantifiers are simply dropped (like in the first three cases from Table 2.3), whereas existential quantifiers result in *Skolem functions*: a formula of the form $\forall x.\exists y.F(x, y)$ expressing that for every x there exists y that satisfies $F(x, y)$ is replaced with $F(x, f(x))$, where $f(x)$ is a *fresh* function which makes this dependence between x and y explicit. In Table 2.3 we apply *Skolemisation* only for the first-order formulas that originate from *inclusion axioms* of form $A \sqsubseteq \exists R.B$, since those are the only formulas containing the existential quantifier positively. Skolem functions $f_A(x)$ introduced in this transformation, must be *unique* for every definition of this form. We make use of this property by indexing $f_A(x)$ with concept name A, meaning that $f_A(x)$ determines the atom A *uniquely*. In symbols, whenever $f_A = g_B$, then $A = B$ (here "=" denotes the syntactic equality).

Proof search in resolution is based on *refutation*. In order to prove that a set of *axioms* S implies a *conjecture* F, resolution tries to obtain a contradiction from $S \cup \{\neg F\}$. When we apply this principle for reasoning problems in description logics, we should (**1**) treat formulas that correspond to TBox and ABox as axioms and (**2**) negate the formula corresponding to a query: see Table 2.4. To derive a contradiction from such hypothesis, a resolution-based prover performs inferences from clauses using special *inference rules* that form a *calculus*. This iterative process of deriving new clauses from old ones is called *saturation*. New clauses are derived until either the *empty clause* $\Box$ is obtained (that is a clause containing no literals), or all possible inferences are performed. In the first case, we say that a *refutation* has been found and so, the initial clause set is *unsatisfiable*, which means that our conjecture F is provable from axioms S. In the letter case, we say that a *saturation* is computed which indicates that the initial clause set is *satisfiable* and therefore,

Table 2.4 A **CNF**-translation for some reasoning problems in description logics

Concept Subsumption:		**CNF** – *translation*	*clause type*
?- $A \sqsubseteq B$	$\neg(\forall x.[A(x) \rightarrow B(x)])$	$A(c)$ $\neg B(c)$	$\mathsf{C1}(A, c)$ $\mathsf{D1}(B, c)$
Instance Problem:		**CNF** – *translation*	*clause type*
?- $a : A$	$\neg A(a)$	$\neg A(a)$	$\mathsf{D1}(A, a)$
?- $(a, b) : R$	$\neg R(a, b)$	$\neg R(a, b)$	$\mathsf{D2}(R, a, b)$

hypothesis F is *not* provable from S.

The *ordered resolution calculus* consists of two inference rules: Ordered Resolution and Ordered Factoring, given in Figure 2.1. For our procedures we will use an additional optional *simplification rule* Elimination of Duplicate Literals. The Ordered Resolution rule allows one to derive a new clause by *resolving* two clauses with complementary literals. For example, clause $\neg A(x) \lor \underline{B(f_A(x))}$ might be *resolved* with clause $\neg \underline{B(x)} \lor C(x)$ producing $\neg A(x) \lor C(f_A(x))$. Whether an inference can be carried out or not, depends on the conditions of the inference rule. The first condition of Ordered Resolution requires that the resolved atoms (which are underlined) must be *unifiable*. The last two conditions require that the resolved atoms must be eligible (written in bold) in the clauses.

Figure 2.1 The ordered resolution calculus with selection and simplification

Ordered Resolution

$$\mathtt{OR}: \frac{C \lor \underline{\boldsymbol{A}}^{\star} \quad D \lor \neg\underline{\boldsymbol{B}}}{C\sigma \lor D\sigma}$$

[*where (i)* $\sigma = \mathrm{mgu}(A, B)$*; (ii)* A *is eligible strictly maximal in the clause* $C \lor A$*, and (iii)* $\neg B$ *is eligible in the clause* $D \lor \neg B$.]

Ordered Factoring

$$\mathtt{OF}: \frac{C \lor \underline{B} \lor \underline{\boldsymbol{A}}}{C\sigma \lor A\sigma}$$

[*where (i)* $\sigma = \mathrm{mgu}(A, B)$*; (ii)* A *is eligible in clause* $C \lor B \lor A$.]

Elimination of Duplicate Literals

$$\mathtt{ED}: \frac{[\![C \lor L \lor L]\!]}{C \lor L}$$

[*where the premise of the rule is deleted*]

Eligible atoms are determined by two *parameters* of the resolution calculus, namely a *selection function* and an *ordering*. These parameters are used as follows. For every clause it is allowed to select one of its *negative* literals. Then resolution must be carried out on one of the selected literals. If no literal is selected in a clause, then this clause must be resolved upon its *maximal* literals, which are determined using the *ordering*. For our purpose, we take an ordering $\succ$ such that

$L_1 \succ L_2$ whenever (***i***) L_1 contain more symbols than L_2 and (***ii***) this property is preserved under substitutions of terms for variables in clauses (more precisely, we use the *Knuth-Bendix ordering* with equal weights for all symbols). A literal L is *(strictly) maximal* in a clause C if there is no other literal L_1 in C such that $L_1 \succ L$ ($L_1 \succeq L$). We will indicate the literals in rules that are required to be strictly maximal by star. For example, it is easy to see that literal $B(f_A(x))$ on which we have resolved the clause $\neg A(x) \vee \underline{B(f_A(x))}$ above, is strictly maximal in this clause. To summarise, a literal is *eligible* in a clause if it is either selected in this clause, or, otherwise, nothing is selected in this clause and this literal is maximal.

Please see an example in Appendix A.1 demonstrating how subsumption and instance relations can be established formally using the ordered resolution calculus. This example shows how to process queries ?- Father $\sqsubseteq$ Parent, ?- Grandfather $\sqsubseteq$ Father and ?- *John* : Grandfather for the terminology given in Table 2.1.

2.3 A Resolution Decision Procedure for $\mathcal{EL}$

Computing *finite* saturations is the key point of using resolution as a decision procedure. The ordered resolution calculus given in Figure 2.1 is *sound* and *refutationally complete*. Soundness ensures that a contradiction (the empty clause) cannot be derived from a *satisfiable* clause set, whereas completeness means that the empty clause is always derivable from every *unsatisfiable* clause set. Soundness and completeness mean that resolution can be used as a *semi-decision procedure* for the full first-order logic. In order to turn resolution into a *decision procedure* for a particular class of formulas, one has to ensure that this procedure *always terminates* for this class. Our goal now, is to describe a resolution strategy according to which inferences from a finite set of clauses for $\mathcal{EL}$ produce only *finitely many* different clauses.

2.3.1 Enforcing Termination

It is not surprising that subsumption or instance relations between concepts and individuals can be *proven* using the ordered resolution calculus. Resolution, as any other *complete* calculus for first-order logic, is (theoretically) capable of proving any conjecture. It is more tricky to prove *anti-subsumption*, i.e., that one concept does *not* subsume the other. For this, one has to show that the correspondent set of clauses is *satisfiable*, that is, the empty clause is *not* derivable from this clause set using resolution inferences. A naïve solution, is to apply all possible inferences and compute a *saturation* of a clause set, and afterwards check, if the empty clause is there. However, the saturation process might not terminate in general. Fortunately, for $\mathcal{EL}$ we are able to give a resolution strategy that is *guaranteed* to produce only *finitely many* clauses.

In Table 2.5 we have listed all possible *types of clauses* that can be produced by resolution from clauses for $\mathcal{EL}$ obtained according to Table 2.3 and Table 2.4. Every bold symbol $\boldsymbol{A}$, $\boldsymbol{R}$, $\boldsymbol{f_A}$, $\boldsymbol{a}$, etc... in these clause types is a *parameter* of a clause type. Parameters can be instantiated with any predicate symbol (= concept name or role name), functional symbol (= Skolem function) or constant (= individual) that occurs in initial clauses resulted from **CNF**-transformation of TBox and ABox. It is only required that the obtained clause should be *meaningful*, i.e., it is not allowed to substitute, say, a functional symbol for $\boldsymbol{A}$, or a concept name for $\boldsymbol{R}$. There are only *finitely many* clauses that can be obtained in this way for a fixed TBox and ABox. Hence, if we prove that every clause derived by resolution from $\mathcal{EL}$-TBox and ABox must be one of them, this would mean that a finite saturation is computed for every input, and so, resolution can decide reasoning problems in $\mathcal{EL}$.

Table 2.5 Clause types for $\mathcal{EL}$

C1 $\underline{\boldsymbol{A}(\boldsymbol{a})}$;	D1 $\neg\underline{\boldsymbol{A}(\boldsymbol{a})}$;	$\bot$ $\square$;
C2 $\underline{\boldsymbol{R}(\boldsymbol{a},\boldsymbol{b})}$;	D2 $\neg\underline{\boldsymbol{R}(\boldsymbol{a},\boldsymbol{b})}$;	
	D3 $\neg\underline{\boldsymbol{A}(\boldsymbol{a})} \vee \boldsymbol{B}(\boldsymbol{b})$;	
C3 $\underline{\boldsymbol{A}(x)}$;		
C4 $\neg\boldsymbol{A}(x) \vee \underline{\boldsymbol{B}(\boldsymbol{f_A}(x))}$;	D4 $\neg\underline{\boldsymbol{A}(x)} \vee \boldsymbol{B}(x)$;	
C5 $\neg\boldsymbol{A}(x) \vee \underline{\boldsymbol{R}(x,\boldsymbol{f_A}(x))}$;	D5 $\neg\underline{\boldsymbol{A}(x)} \vee \neg\boldsymbol{B}(x) \vee \boldsymbol{C}(x)$;	
	D6 $\neg\underline{\boldsymbol{R}(x,y)} \vee \neg\boldsymbol{B}(y) \vee \boldsymbol{A}(x)$;	
	D7 $\neg\boldsymbol{A}(x) \vee \neg\underline{\boldsymbol{B}(\boldsymbol{f_A}(x))} \vee \boldsymbol{C}(x)$;	
	D8 $\neg\boldsymbol{A}(x) \vee \neg\underline{\boldsymbol{B}(\boldsymbol{f_A}(x))} \vee \boldsymbol{C}(\boldsymbol{f_A}(x))$;	

In order prove that resolution for $\mathcal{EL}$ can produce only clauses from Table 2.5, we demonstrate that (**1**) every clause resulted from **CNF** translation for $\mathcal{EL}$ has one of these types, and (**2**) the conclusion of every resolution inference from clauses of these types is again a clause of these types. The first property is easy to observe: in the last column of Table 2.3 and Table 2.4 we have already indicated the types of the clauses resulted from a **CNF**-translation. For showing the second property, we enumerate all possible inferences between clauses in Table 2.5.

The Ordered Factoring rule cannot be applied to any of these clauses, because they contain at most one positive literal. Ordered Resolution inferences are only possible between clauses C1 – C5 with a positive eligible literal, and clauses D1 – D8 with a negative eligible literal. For conciseness, we describe resolution inferences only between clauses C3 – C5 and D4 – D6, and later, sketch inferences between other clauses. Clauses C3 – C5 and D4 – D6 (below the dashed line) originate from $\mathcal{EL}$-TBoxes and, in certain sense, correspond to purely terminological reasoning:

(**C3**) A clause of type C3 can be resolved with a clause of type D4, D5, D7 or D8. It is not possible to resolve a clause of type C3 with a clause of type D6 since the number of arguments in eligible literals does not match. The remaining inferences are written below:

$\mathtt{OR}[\mathsf{C3};\mathsf{D4}]$: $\underline{A(x)}$; $\neg\underline{A(x)} \vee B(x) \vdash B(x)$, which is a clause of type $\mathsf{C3}$;

$\mathtt{OR}[\mathsf{C3};\mathsf{D5}]$: $\underline{A(x)}$; $\neg\underline{A(x)} \vee \neg B(x) \vee C(x) \ \vdash\ \neg B(x) \vee C(x)$, which is a clause of type $\mathsf{D4}$;

$\mathtt{OR}[\mathsf{C3};\mathsf{D7}]$: $\underline{A(x)}$; $\neg B(x) \vee \neg\underline{A(f_B(x))} \vee C(x) \ \vdash\ \neg B(x) \vee C(x)$, which is again a clause of type $\mathsf{D4}$. Note that we have renamed clause parameters in the second clause to match the unified expressions. Clause parameters are, essentially, treated as *meta-variables*;

$\mathtt{OR}[\mathsf{C3};\mathsf{D8}]$: $\underline{A(x)}$; $\neg B(x) \vee \neg\underline{A(f_B(x))} \vee C(f_B(x)) \ \vdash\ \neg B(x) \vee C(f_B(x))$, $\Rightarrow \mathsf{C4}$;

($\mathsf{C}\mathbf{4}$) A clause of type $\mathsf{C4}$ can be resolved with a clauses of types $\mathsf{D4}$, $\mathsf{D5}$, $\mathsf{D7}$, $\mathsf{D8}$:

$\mathtt{OR}[\mathsf{C4};\mathsf{D4}]$: $\neg A(x) \vee \underline{B(f_A(x))}$; $\neg\underline{B(x)} \vee C(x) \ \vdash\ \neg A(x) \vee C(f_A(x))$, which is a clause of type $\mathsf{C4}$;

$\mathtt{OR}[\mathsf{C4};\mathsf{D5}]$: $\neg A(x) \vee \underline{B(f_A(x))}$; $\neg\underline{B(x)} \vee \neg C(x) \vee D(x) \ \vdash\ \neg A(x) \vee \neg C(f_A(x)) \vee D(f_A(x))$, which is a clause of type $\mathsf{D8}$;

$\mathtt{OR}[\mathsf{C4};\mathsf{D7}]$: $\neg A(x) \vee \underline{B(f_A(x))}$; $\neg A(x) \vee \neg\underline{B(f_A(x))} \vee C(x) \ \vdash\ \neg A(x) \vee \neg A(x) \vee C(x) \ \vdash\ \neg A(x) \vee C(x)$, which is a clause of type $\mathsf{D4}$. Note that we have used a property according to which functional symbol f_A uniquely determines atom A in its index. Hence, if clauses of the above form can be resolved, then they must have the same negative literal $\neg A(x)$ and so, duplicate occurrences of this literal in the conclusion of this rule can be removed;

$\mathtt{OR}[\mathsf{C4};\mathsf{D8}]$: $\neg A(x) \vee \underline{B(f_A(x))}$; $\neg A(x) \vee \neg\underline{B(f_A(x))} \vee C(f_A(x)) \ \vdash\ \neg A(x) \vee \neg A(x) \vee C(f_A(x)) \ \vdash\ \neg A(x) \vee C(f_A(x))$, which is a clause of type $\mathsf{C4}$. The same argument as in the previous case allows us to simplify the conclusion of this inference;

($\mathsf{C}\mathbf{5}$) A clause of type $\mathsf{C5}$ can be resolved only with a clause of type $\mathsf{D6}$, since otherwise the number of arguments in the unified literals does not match:

$\mathtt{OR}[\mathsf{C5};\mathsf{D6}]$: $\neg A(x) \vee \underline{R(x, f_A(x))}$; $\neg\underline{R(x, y)} \vee \neg B(y) \vee C(x) \ \vdash\ \neg A(x) \vee \neg B(f_A(x)) \vee C(x)$, which is a clause of type $\mathsf{D7}$.

In Table 2.6 we have summarised all possible inferences between clauses from Table 2.5. Inferences $\mathsf{T1}$ – $\mathsf{T9}$ corresponding to terminological reasoning have been considered in more details above. Inferences $\mathsf{A1}$ – $\mathsf{A8}$ correspond to reasoning with individuals, i.e., they involve clauses that result from translation of ABox-es, queries or are derived from such clauses. Note that no inference $\mathsf{A1}$ – $\mathsf{A8}$ uses clause types $\mathsf{C4}$, $\mathsf{C5}$, $\mathsf{D7}$ or $\mathsf{D8}$. So, clauses of the remaining types: $\mathsf{C3}$ and $\mathsf{D4}$ – $\mathsf{D6}$ can be seen as *extended taxonomy* of concepts: only these clauses computed for a TBox, are needed to answer queries about individuals.

Table 2.6 Summary for inferences between clauses for $\mathcal{EL}$

A1.	OR[C1; D1]: $\underline{A(a)};\ \neg\underline{A(a)} \vdash \Box$	: $\bot$
A2.	OR[C1; D3]: $\underline{A(a)};\ \neg\underline{A(a)} \vee B(b) \vdash B(b)$	: C1
A3.	OR[C1; D4]: $\underline{A(a)};\ \neg\underline{A(x)} \vee B(x) \vdash B(a)$	: C1
A4.	OR[C1; D5]: $\underline{A(a)};\ \neg\underline{A(x)} \vee \neg B(x) \vee C(x) \vdash \neg B(a) \vee C(a)$	: D3
A5.	OR[C2; D2]: $\underline{R(a,b)};\ \neg\underline{R(a,b)} \vdash \Box$	: $\bot$
A6.	OR[C2; D6]: $\underline{R(a,b)};\ \neg\underline{R(x,y)} \vee \neg B(y) \vee A(x) \vdash \neg B(b) \vee A(a)$	: D3
A7.	OR[C3; D1]: $\underline{A(x)};\ \neg\underline{A(a)} \vdash \Box$	: $\bot$
A8.	OR[C3; D3]: $\underline{A(x)};\ \neg\underline{A(a)} \vee B(b) \vdash B(b)$	: C1
T1.	OR[C3; D4]: $\underline{A(x)};\ \neg\underline{A(x)} \vee B(x) \vdash B(x)$	: C3
T2.	OR[C3; D5]: $\underline{A(x)};\ \neg\underline{A(x)} \vee \neg B(x) \vee C(x) \vdash \neg B(x) \vee C(x)$	: D4
T3.	OR[C3; D7]: $\underline{A(x)};\ \neg B(x) \vee \neg\underline{A(f_B(x))} \vee C(x) \vdash \neg B(x) \vee C(x)$	: D4
T4.	OR[C3; D8]: $\underline{A(x)};\ \neg B(x) \vee \neg\underline{A(f_B(x))} \vee C(f_B(x)) \vdash \neg B(x) \vee C(f_B(x))$	: C4
T5.	OR[C4; D4]: $\neg A(x) \vee \underline{B(f_A(x))};\ \neg\underline{B(x)} \vee C(x) \vdash \neg A(x) \vee C(f_A(x))$	: C4
T6.	OR[C4; D5]: $\neg A(x) \vee \underline{B(f_A(x))};\ \neg\underline{B(x)} \vee \neg C(x) \vee D(x) \vdash$ $\vdash \neg A(x) \vee \neg C(f_A(x)) \vee D(f_A(x))$	: D8
T7.	OR[C4; D7]: $\neg A(x) \vee \underline{B(f_A(x))};\ \neg A(x) \vee \neg\underline{B(f_A(x))} \vee C(x) \vdash$ $\vdash \neg A(x) \vee \neg A(x) \vee C(x) \vdash \neg A(x) \vee C(x)$	: D4
T8.	OR[C4; D8]: $\neg A(x) \vee \underline{B(f_A(x))};\ \neg A(x) \vee \neg\underline{B(f_A(x))} \vee C(f_A(x)) \vdash$ $\vdash \neg A(x) \vee \neg A(x) \vee C(f_A(x)) \vdash \neg A(x) \vee C(f_A(x))$	: C4
T9.	OR[C5; D6]: $\neg A(x) \vee \underline{R(x, f_A(x))};\ \neg\underline{R(x,y)} \vee \neg B(y) \vee C(x) \vdash$ $\vdash \neg A(x) \vee \neg B(f_A(x)) \vee C(x)$	: D7

2.3.2 Making It Simple

In the previous section we have demonstrated how reasoning problems in $\mathcal{EL}$ can be solved using the ordered resolution calculus. This means that any resolution theorem prover can, in principle, be used as a complete $\mathcal{EL}$-classifier. It comes to our surprise, that no theorem prover is actually required to *implement* our decision procedure. If we look carefully in Table 2.6, we may notice that all inferences T1 – T9 and A1 – A8 can be formulated as very simple instructions. For example, inference T1 says: "*given a clause of type* C3 *with parameter* A *and a clause of type* D4 *with parameters* A *and* B, *produce a clause of type* C3 *with parameter* B". These instructions can be written in a form of *datalog rules* (*Horn clauses* with no functional symbols, which are usually written right-to-left), where clause types are treated as atoms and clause parameters are treated as variables: see Table 2.7 (please, ignore the underlying and bold marks for now).

So, the overall procedure for reasoning in $\mathcal{EL}$ can be seen as follows. Given

Table 2.7 A datalog program for reasoning problems in $\mathcal{EL}$

A1.	$\bot$	$\leftarrow \mathsf{C1}(\boldsymbol{A}, \boldsymbol{a}),\ \underline{\mathsf{D1}(\boldsymbol{A}, \boldsymbol{a})}.$
A2.	$\mathsf{C1}(B, b)$	$\leftarrow \mathsf{C1}(\boldsymbol{A}, \boldsymbol{a}),\ \dashuline{\mathsf{D3}(\boldsymbol{A}, B, \boldsymbol{a}, b)}.$
A3.	$\mathsf{C1}(B, a)$	$\leftarrow \mathsf{C1}(\boldsymbol{A}, a),\ \mathsf{D4}(\boldsymbol{A}, B).$
A4.	$\mathsf{D3}(B, C, a, a)$	$\leftarrow \mathsf{C1}(\boldsymbol{A}, a),\ \underline{\mathsf{D5}(\boldsymbol{A}, B, C)}.$
A5.	$\bot$	$\leftarrow \mathsf{C2}(\boldsymbol{R}, \boldsymbol{a}, \boldsymbol{b}),\ \underline{\mathsf{D2}(\boldsymbol{R}, \boldsymbol{a}, \boldsymbol{b})}.$
A6.	$\mathsf{D3}(B, A, b, a)$	$\leftarrow \underline{\mathsf{C2}(\boldsymbol{R}, a, b)},\ \underline{\mathsf{D6}(\boldsymbol{R}, B, A)}.$
A7.	$\bot$	$\leftarrow \mathsf{C3}(\boldsymbol{A}),\ \underline{\mathsf{D1}(\boldsymbol{A}, a)}.$
A8.	$\mathsf{C1}(B, b)$	$\leftarrow \mathsf{C3}(\boldsymbol{A}),\ \dashuline{\mathsf{D3}(\boldsymbol{A}, B, a, b)}.$
T1.	$\mathsf{C3}(B)$	$\leftarrow \mathsf{C3}(\boldsymbol{A}),\ \mathsf{D4}(\boldsymbol{A}, B).$
T2.	$\mathsf{D4}(B, C)$	$\leftarrow \mathsf{C3}(\boldsymbol{A}),\ \underline{\mathsf{D5}(\boldsymbol{A}, B, C)}.$
T3.	$\mathsf{D4}(B, C)$	$\leftarrow \mathsf{C3}(\boldsymbol{A}),\ \dashuline{\mathsf{D7}(B, \boldsymbol{A}, C, f_B)}.$
T4.	$\mathsf{C4}(B, C, f_B)$	$\leftarrow \mathsf{C3}(\boldsymbol{A}),\ \dashuline{\mathsf{D8}(B, \boldsymbol{A}, C, f_B)}.$
T5.	$\mathsf{C4}(A, C, f_A)$	$\leftarrow \mathsf{C4}(A, \boldsymbol{B}, f_A),\ \mathsf{D4}(\boldsymbol{B}, C).$
T6.	$\mathsf{D8}(A, C, D, f_A)$	$\leftarrow \mathsf{C4}(A, \boldsymbol{B}, f_A),\ \underline{\mathsf{D5}(\boldsymbol{B}, C, D)}.$
T7.	$\mathsf{D4}(A, C)$	$\leftarrow \mathsf{C4}(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{f_A}),\ \dashuline{\mathsf{D7}(\boldsymbol{A}, \boldsymbol{B}, C, \boldsymbol{f_A})}.$
T8.	$\mathsf{C4}(A, C, f_A)$	$\leftarrow \mathsf{C4}(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{f_A}),\ \dashuline{\mathsf{D8}(\boldsymbol{A}, \boldsymbol{B}, C, \boldsymbol{f_A})}.$
T9.	$\mathsf{D7}(A, B, C, f_A)$	$\leftarrow \mathsf{C5}(A, \boldsymbol{R}, f_A),\ \underline{\mathsf{D6}(\boldsymbol{R}, B, C)}.$

a terminology TBox, a set of assertions ABox and a query Q, one translates this input data to a set of *ground facts* according to Table 2.3 and Table 2.4 (now, please pay attention to their last columns). After that, one computes a *deductive closure* of this database, by exhaustively applying all rules from Table 2.7. The query Q is answered *positively* if $\bot$ is obtained in the result, and otherwise, the query is evaluated *negatively*.

For example, our terminology of human relations (2.3) with correspondent assertions and queries is translated to the set of ground facts given in Table 2.8. Please, take a look at Appendix A.2, where we have demonstrated how to compute a deductive closure for this set of ground facts, and evaluate the queries.

A datalog program in Table 2.7 can be seen as a *compiled representation* of our resolution decision procedure for $\mathcal{EL}$, in the sense that we have specialised a resolution theorem prover on our particular class of clauses $\mathsf{C1}$ – $\mathsf{C5}$, $\mathsf{D1}$ – $\mathsf{D8}$ and classified inferences between them. Hence a system performing inferences according to our scheme, should not spend time on unification attempts and computation of inferences. Moreover, evaluation of a datalog program need not necessary proceed by *saturation* of the input database, in other words, *bottom-up*. And indeed, most of deductive database systems evaluate queries *top-down* in *depth-first* manner, starting from the goal and applying SLD-resolution with datalog clauses. This makes query evaluation goal-oriented and *space efficient*, since backtracking is employed.

The rules for *implication sets*, also called *completion rules*, proposed in recent

Table 2.8 A datalog translation for terminology of human relations

TBox-*definition*	$\dashrightarrow$	*Clause Types*		
$\mathsf{Man} \doteq \mathsf{Human} \sqcap \mathsf{Male}$	$\dashrightarrow$	$\mathsf{D4(Man, Human)}$,	$\mathsf{D4(Man, Male)}$,	$\mathsf{D5(Human, Male, Man)}$
$\mathsf{A} \doteq \exists \mathsf{has\text{-}child.Human}$	$\dashrightarrow$	$\mathsf{C5(A, has\text{-}child}, \mathtt{f})$,	$\mathsf{C4(A, Human}, \mathtt{f})$,	$\mathsf{D6(has\text{-}child, Human, A)}$
$\mathsf{Parent} \doteq \mathsf{Human} \sqcap \mathsf{A}$	$\dashrightarrow$	$\mathsf{D4(Parent, Human)}$,	$\mathsf{D4(Parent, A)}$,	$\mathsf{D5(Human, A, Parent)}$
$\mathsf{Father} \doteq \mathsf{Man} \sqcap \mathsf{A}$	$\dashrightarrow$	$\mathsf{D4(Father, Man)}$,	$\mathsf{D4(Father, A)}$,	$\mathsf{D5(Man, A, Father)}$
$\mathsf{B} \doteq \exists \mathsf{has\text{-}child.Parent}$	$\dashrightarrow$	$\mathsf{C5(B, has\text{-}child, g)}$,	$\mathsf{C4(B, Parent, g)}$,	$\mathsf{D6(has\text{-}child, Parent, B)}$
$\mathsf{Grandfather} \doteq \mathsf{Man} \sqcap \mathsf{B}$	$\dashrightarrow$	$\mathsf{D4(Grandfather, Man)}$,	$\mathsf{D4(Grandfather, B)}$,	$\mathsf{D5(Man, B, Grandfather)}$
ABox-*definition*	$\dashrightarrow$	*Clause Types*		
John : Man	$\dashrightarrow$	$\mathsf{C1(Man}, \mathit{John})$		
Bill : Father	$\dashrightarrow$	$\mathsf{C1(Father}, \mathit{Bill})$		
$(\mathit{John}, \mathit{Bill})$: $\mathsf{has\text{-}child}$	$\dashrightarrow$	$\mathsf{C2(has\text{-}child}, \mathit{John}, \mathit{Bill})$		
Query	$\dashrightarrow$	*Clause Types*		
?- $\mathsf{Grandfather} \sqsubseteq \mathsf{Father}$	$\dashrightarrow$	$\mathsf{C1(Grandfather, c)}$	$\mathsf{D1(Father, c)}$	
?- *John* : $\mathsf{Grandfather}$	$\dashrightarrow$	$\mathsf{D1(Grandfather}, \mathit{John})$		

papers [Brandt, 2004*a*; Baader et al., 2005] for reasoning in $\mathcal{EL}$ and its extensions, can be also naturally formulated as datalog programs. For example, a subset of rules **CR1** – **CR4** from [Baader et al., 2005], which corresponds to terminological reasoning in $\mathcal{EL}$, can be seen as the following datalog program:

$$\begin{array}{lll} \textbf{IR1} & C \in s(C) & \leftarrow . \\ \textbf{IR2} & \top \in s(C) & \leftarrow . \\ \textbf{CR1} & D \in s(C) & \leftarrow C' \in s(C), \quad C' \sqsubseteq D. \\ \textbf{CR2} & D \in s(C) & \leftarrow C_1 \in s(C), \quad C_2 \in s(C), \quad C_1 \sqcap C_2 \sqsubseteq D. \\ \textbf{CR3} & (C, D) \in r(R) & \leftarrow C' \in s(C), \quad C' \sqsubseteq \exists R.D. \\ \textbf{CR4} & E \in s(C) & \leftarrow (C, D) \in r(R), \quad D' \in s(D), \quad \exists R.D' \sqsubseteq E. \end{array} \tag{2.4}$$

where "$D \in s(C)$", "$(C, D) \in r(R)$", "$C \sqsubseteq D$", "$C_1 \sqcap C_2 \sqsubseteq D$", "$C \sqsubseteq \exists R.D$" and "$\exists R.D \sqsubseteq E$" shall be thought of as *datalog atoms* with correspondent parameters-variables. For example, "$D \in s(C)$" is an atom having two variables C and D, which expresses that concept D *subsumes* concept C. Using these rules, the subsumption relation for $\mathcal{EL}$ can be computed explicitly and *directly* from simplified terminological axioms.

There is, however, an important difference between our approach and the approach of Brandt [2004*a*], Baader et al. [2005]. According to our method, we do not *postulate* the completion rules, but we rather *derive* them from the semantics of constructors using the ordered resolution calculus. This has an advantage that the proof of correctness for the resulted algorithm comes “for free”. I.e., the only thing we rely on, is the completeness of general ordered resolution calculus. This is not a big issue for the particular logic $\mathcal{EL}$, but it will become important for extensions of $\mathcal{EL}$ for which correctness proofs are more involved.

To prove a correctness of an algorithm, one should generally prove three things: (**1**) *soundness*, (**2**) *completeness* and (**3**) *termination* of the procedure. For us, *soundness* and – the most difficult part – *completeness*, are implied by soundness and completeness of ordered resolution. What usually remains to be shown for resolution decision procedures, is *termination*, which is obvious in our case, since a deductive closure under datalog rules can be always computed in finite time.

The usage of deductive database systems for reasoning in $\mathcal{EL}$, has another advantage, namely, that one can utilise their native capabilities of query evaluation. Consider, for example, the *retrieval problem*: ?- $x : C$ asking for all instances of a concept C (w.l.o.g. we assume that C is a concept name). We know, that in order to check whether an individual a is an instance of C (which is the *instance problem*), we need to derive $\bot$ from ground datalog atoms for TBox, ABox, and atom $\mathsf{D1}(C, a)$ (see Table 2.4). We may assume that this is the only atom of form $\mathsf{D1}(*, *)$ and there is no atom of form $\mathsf{D2}(*, *, *)$, since we process only one query at a time. Atom $\bot$ can be derived only by rules A1, A5 and A7, each of which contains either atom $\mathsf{D1}(*, *)$ or $\mathsf{D2}(*, *, *)$ in the *body*. So, for proving instance $a : C$, we have to derive either $\mathsf{C1}(C, a)$ or $\mathsf{C3}(A)$. This can be reformulated explicitly using two additional rules I1 and I2 given in Table 2.9. Now the *retrieval problem*: ?- $x : C$ can be solved

Table 2.9 An extension of the datalog program for instance and retrieval problems in $\mathcal{EL}$

I1.	$\mathsf{Instances}(a, C)$	$\leftarrow \mathsf{C1}(C, a);.$
I2.	$\mathsf{Instances}(a, C)$	$\leftarrow \mathsf{C3}(C);.$
R1.	$\mathsf{Relations}(a, R, b)$	$\leftarrow \mathsf{C2}(R, a, b);.$

using a datalog query ?- $\mathsf{Instances}(X, C)$. One can compute other queries for ABox and TBox, for example ?- $\mathsf{Instances}(a, X)$ will print all concept names for individual a. Similarly, an extension of our datalog program with rule R1 from Table 2.9 allows one to query role relations between individuals.

Using such *transformation of queries*, we can *derive* rules for the explicit *subsumption relation* similar to those in (2.4). Please see Appendix A.3 to find details of how we derive datalog rules in Table 2.10 for computation of the explicit subsumption relation $\mathsf{subsumes}(B, A)$. Using this extension, we can query TBox, say by asking ?- $\mathsf{subsumes}(B, X)$ for a list of all concept names subsumed by B.

Table 2.10 A datalog program for classification of $\mathcal{EL}$-terminologies

S0	$\mathsf{sb}(A, A)$	$\leftarrow$	.
S1	$\mathsf{subsumes}(B, A)$	$\leftarrow$	$\mathsf{C3}(B)$.
S2	$\mathsf{subsumes}(B, A)$	$\leftarrow$	$\mathsf{sb}(B, A)$.
S3	$\mathsf{sb}(E, A)$	$\leftarrow$	$\mathsf{sb}(\boldsymbol{C}, A),\ \mathsf{D4}(\boldsymbol{C}, E)$.
S4	$\mathsf{sb}(E, A)$	$\leftarrow$	$\mathsf{sb}(\boldsymbol{C}, A),\ \mathsf{sb}(\boldsymbol{D}, A),\ \underline{\mathsf{D5}(\boldsymbol{D}, \boldsymbol{C}, E)}$.
S5	$\mathsf{sb}(E, A)$	$\leftarrow$	$\mathsf{C3}(\boldsymbol{C}),\ \mathsf{sb}(\boldsymbol{D}, A),\ \underline{\mathsf{D5}(\boldsymbol{D}, \boldsymbol{C}, E)}$.

2.3.3 Complexity Analysis

It is well-known that a datalog query can be evaluated in polynomial time in the size of the initial database (see, for instance [Dantsin, Eiter, Gottlob & Voronkov, 2001]). This means that the reasoning problems for $\mathcal{EL}$ can be decided in polynomial time using our reduction to datalog. In this section we demonstrate how to extract more exact complexity bounds for our procedures.

Given a datalog program P and a database of facts D, a closure $P(\mathrm{D})$ of D under rules of P, can be computed in time $O(|\mathrm{D}|^k)$, where k is the maximal number of different variables in each rule from P, and $|\mathrm{D}|$ is the size of D. Applying this estimation to the program in Table 2.7, we conclude that the considered reasoning problems for $\mathcal{EL}$ can be decided in time $O(n^5)$, where n bounds the size of TBox and ABox, since every rule in our program contains at most 5 variables. This is not a very exciting result. Now we demonstrate how this estimation can be improved to $O(n^3)$ using more careful calculations.

We will use a well-known technique from optimisation of logical programs and estimation of their running times, which is elegantly formulated in [McAllester, 2002]. One of the theorems from this paper says that *a deductive closure $P(D)$ of a database D by a datalog program P with range-restricted rules can be computed in time $O(|D| + |PF_P(P(D))|)$*, where $\mathrm{PF}_P(\mathrm{D}')$ is the set of *prefix firings* of program P w.r.t. a database D'. Let us explain what a *range restricted* rule is and how to compute *prefix firings* for a program.

A datalog rule is called *range-restricted* if every variable occurring in the head of this rule, also occurs in the body of this rule. It is easy to observe that all rules in Table 2.7 are range restricted. A *prefix firing* for a datalog rule $B \leftarrow A_1, \ldots, A_n$ w.r.t. a database D' is a vector $A'_1, \ldots, A'_k$ with $A'_i \in \mathrm{D}'$, $1 \leq i \leq k \leq n$, which can be matched to first k atoms in the body of this rule (see [McAllester, 2002] for more details). Intuitively, a prefix firing of a rule is a partial evaluation of this rule for first k atoms in the body. A set of prefix firings $\mathrm{PF}_P(\mathrm{D}')$ for a datalog program w.r.t. a database D' is the union of all prefix firings for all rules in P w.r.t. D'.

The number of prefix firings for a rule, clearly depends on the order of atoms in the body of this rule. Let us look more carefully at the rules in Table 2.7. We may notice that some atoms may never be produced as conclusions of these rules, in particular atoms of forms $\mathsf{C2}(*,*,*)$, $\mathsf{D1}(*,*)$, $\mathsf{D2}(*,*,*)$, $\mathsf{D5}(*,*,*)$ and $\mathsf{D6}(*,*,*)$. Now take a look at the rules that contain these atoms in the body (underlined with a single line), say rule $\mathsf{T6}$. Let us estimate the number of prefix firings of this rule w.r.t. the closure $P(\mathrm{D})$ of the initial database D, when we reverse the order the atoms in the body of $\mathsf{T6}$:

$$\mathsf{T6}.\quad \mathsf{D8}(A,C,D,f_A) \leftarrow \underline{\mathsf{D5}(\boldsymbol{B},C,D)},\ \mathsf{C4}(A,\boldsymbol{B},f_A). \tag{2.5}$$

The number of prefix firings for the first atom $\mathsf{D5}(\boldsymbol{B},C,D)$ is bounded by $|\mathrm{D}|$, since atoms of this type cannot be produced by any inference. In order to estimate the number of prefix firings for both atoms, note that each firing for the first atom grounds the *shared variable* $\boldsymbol{B}$ contained in both atoms of this rule. Hence, each candidate for the second atom is determined by the values for the remaining variables A and f_A. However, we know that every value of f_A uniquely determines the value of A. Hence, there are at most $|\mathrm{D}|$ choices for the second atom left, since there are at most $|\mathrm{D}|$ symbols f_A. Consequently, we have at most $|\mathrm{D}|$ prefix firings for the first atom and for each of them, at most $|\mathrm{D}|$ firings for the second atom, so the total number of prefix firings for this rule is at most $|\mathrm{D}| + |\mathrm{D}|^2$.

A similar analysis can be carried out for rules $\mathsf{A1}$, $\mathsf{A4}$ – $\mathsf{A7}$, $\mathsf{T2}$, $\mathsf{T6}$ and $\mathsf{T9}$, which shows that there are at most $O(|\mathrm{D}|^2)$ prefix firing for these rules. To estimate the number of prefix firings for other inferences, we note that all above rules produce at most $O(|\mathrm{D}|^2)$ different conclusions. In particular, since atoms of form $\mathsf{D8}(*,*,*,*)$ can be produced only by rule $\mathsf{T6}$, there are at most quadratically many of these in the closure $P(\mathrm{D})$ of our database D. So, we can estimate the number of prefix firings in rules that contain such atoms in the body (we underline them with a dashed line). For example, consider rule $\mathsf{T8}$:

$$\mathsf{T8}.\quad \mathsf{C4}(A,C,f_A) \leftarrow \dashuline{\mathsf{D8}(\boldsymbol{A},\boldsymbol{B},C,\boldsymbol{f_A})},\ \mathsf{C4}(\boldsymbol{A},\boldsymbol{B},\boldsymbol{f_A}). \tag{2.6}$$

As we have figured out, the number of prefix firings for the first atom is at most $O(|\mathrm{D}|^2)$. However, for each choice of the first atom there is at most one choice for the second atom, since all variables of this rule are contained in the first atom. To conclude, there are at most $O(|\mathrm{D}|^2)$ prefix firings for this rule. The same can be shown for rules $\mathsf{A2}$, $\mathsf{A8}$, $\mathsf{T3}$, $\mathsf{T4}$, $\mathsf{T7}$ and $\mathsf{T8}$.

The remaining rules from Table 2.7, like $\mathsf{T5}$ cause a cubic complexity:

$$\mathsf{T5}.\quad \mathsf{C4}(A,C,f_A) \leftarrow \mathsf{C4}(A,\boldsymbol{B},f_A),\ \mathsf{D4}(\boldsymbol{B},C). \tag{2.7}$$

For any order of the atoms in the body of this rule, we have $O(|\mathrm{D}|^2)$ prefix firings for the first atom and one variable that is *not* bounded in the second atoms for which we have at most $O(|\mathrm{D}|)$ choices. Hence, the total number of prefix firings for this rule is bounded by $O(|\mathrm{D}|^3)$.

Applying the theorem from [McAllester, 2002] formulated in the beginning of this section, we conclude that the closure $P(\mathrm{D})$ of our initial database D under the program from Table 2.7 can be computed in cubic time in $|\mathrm{D}|$. Consequently, all our reasoning problems for $\mathcal{EL}$ can be decided in a *cubic time* in the size of TBox and ABox. In fact, computation of *all* instances and subsumption relations can be done in cubic time as well, if we consider the extension of our program with rules from Table 2.9 and Table 2.10. Although the number of prefix firings for these rules can be similarly shown to be at most cubic, we cannot directly apply the above theorem, since not all rules are *range-restricted*: each rule I2, S0 and S1 has a variable in the head that does not occur in the body. However, this can be easily fixed since these variables must be instantiated with either a *concept name* or an *individual name* from the initial database, and we can assume that those are simply listed as ground facts ConceptName(Father), or Individual(*John*). Using these additional "dummy" atoms, the above rules can be modified to range-restricted rules as follows:

$$\begin{array}{lll} \mathsf{I2}' & \mathsf{Instances}(a, C) & \leftarrow \mathsf{C3}(C),\ \mathsf{Individual}(a). \\ \mathsf{S0}' & \mathsf{sb}(A, A) & \leftarrow \mathsf{ConceptName}(A). \\ \mathsf{S1}' & \mathsf{subsumes}(B, A) & \leftarrow \mathsf{C3}(B),\ \mathsf{ConceptName}(A). \end{array} \qquad (2.8)$$

Can one do better?

The *cubic time* algorithm for reasoning in $\mathcal{EL}$ that we have presented, is sufficient for a quite efficient prototypic implementation (see section 2.6). However, there is an natural desire to improve this algorithm somehow, since there are only few rules that cause the cubic complexity. At the moment, we do not see how to lower down this complexity.

It seems to be not clear, whether subsumption of concepts w.r.t. $\mathcal{EL}$-terminology can be decided in *quadratic time*. Baader [2002] has demonstrated that subsumption w.r.t. *normalised* TBoxes can be solved in quadratic time using a reduction to a graph simulation problem. A TBox is *in normal form* if it contains only definitions of form $A \doteq B_1 \sqcap \cdots \sqcap B_k \sqcap \exists R_1.A_1 \sqcap \cdots \sqcap \exists R_l.A_l$, where (***i***) each B_i and A_j with $1 \leq i \leq k$ and $1 \leq j \leq l$ is a *concept name*, and (***ii***) every B_i with $1 \leq i \leq k$ is a *primitive concept*, i.e., there is no definition for it in TBox. Baader [2002] has described a transformation that *normalises* every TBox. However, in contrast to *simplification* of TBoxes (see section 2.2), which can be done in linear time, normalisation process might yield a quadratic blowup in the size of TBox, which can be demonstrated in a simple example below:

$$\begin{array}{rl} \mathsf{A}_1 \doteq \mathsf{B} \sqcap \mathsf{C}_1 \\ \cdots\cdots \\ \mathsf{A}_n \doteq \mathsf{B} \sqcap \mathsf{C}_n \\ \mathsf{B} \doteq \exists \mathsf{R}_1.\mathsf{B}_1 \sqcap \cdots \sqcap \exists \mathsf{R}_n.\mathsf{B}_n \end{array} \quad \Longrightarrow \quad \begin{array}{l} \mathsf{A}_1 \doteq \exists \mathsf{R}_1.\mathsf{B}_1 \sqcap \cdots \sqcap \exists \mathsf{R}_n.\mathsf{B}_n \sqcap \mathsf{C}_1 \\ \mathsf{A}_2 \doteq \exists \mathsf{R}_1.\mathsf{B}_1 \sqcap \cdots \sqcap \exists \mathsf{R}_n.\mathsf{B}_n \sqcap \mathsf{C}_2 \\ \cdots\cdots \\ \mathsf{A}_n \doteq \exists \mathsf{R}_1.\mathsf{B}_1 \sqcap \cdots \sqcap \exists \mathsf{R}_n.\mathsf{B}_n \sqcap \mathsf{C}_n \end{array} \tag{2.9}$$

Hence, a direct complexity estimation of the procedure described in [Baader, 2002], seems to give only $O(n^4)$-time algorithm for checking subsumption in $\mathcal{EL}$. How to avoid this blowup and to find a quadratic time algorithm for subsumption in $\mathcal{EL}$, is one of the challenges for the future work.

2.4 Extensions of DL $\mathcal{EL}$

Although $\mathcal{EL}$ is a relatively simple description logic, it can be already applied for reasoning in existing and widely used commercial terminology SNOMED [Spackman, 2001]. However it is always desirable to identify some other commonly used constructors, adding which would preserve tractability of reasoning. In this section we apply our method to obtain *polynomial* algorithms for some known and new extensions of $\mathcal{EL}$. We also try to explain this nice computational behaviour of $\mathcal{EL}$ and its extensions from resolution point of view.

2.4.1 GCIs, Bottom Concept and Extended Role Hierarchies

Brandt [2004*a*] has observed that reasoning in $\mathcal{EL}$ remains polynomial even with *general inclusion axioms* (short GCIs) of form $C \sqsubseteq D$, where C and D might be arbitrary concepts. Surprisingly, we get this result for free, since we haven't really used that our TBox consists *only* of definitions.[5]

In presence of GCIs, one can use an additional *bottom concept* $\bot$, which intuitively denotes the empty set. This concept, also called the *empty concept*, is not needed, if $\mathcal{EL}$-TBox-es contain *only* definitions.[6]

By allowing GCIs and the empty concept, we definitely extend the expressive power of $\mathcal{EL}$. For example, let us define concepts Woman, Mother and $\mathsf{Grandmother}$ analogously to Man, Father and $\mathsf{Grandfather}$. Then we can express that concepts

[5] Recall that we replace each definition $A \doteq C$ with two inclusion axioms $A \sqsubseteq C$ and $C \sqsubseteq A$

[6] Indeed, $\bot$ can be eliminated from such TBox by a sequence of simple transformations: $A \sqcap \bot \Rightarrow \bot$, $\exists R.\bot \Rightarrow \bot$ and TBox $\cup$ $(A \doteq \bot) \Rightarrow$ TBox$[A/\bot]$

Male and Female are disjoint:

$$\begin{array}{rlr}
\mathsf{Woman} \doteq \mathsf{Human} \sqcap \mathsf{Female} & & \mathsf{Male} \sqcap \mathsf{Female} \sqsubseteq \bot \\
\mathsf{Mother} \doteq \mathsf{Woman} \sqcap \mathsf{Parent} & & \exists \mathsf{has\text{-}child}.\mathsf{Human} \sqsubseteq \mathsf{Human} \\
\mathsf{Grandmother} \doteq \mathsf{Woman} \sqcap \exists \mathsf{has\text{-}child}.\mathsf{Parent} & & \\
\hdashline
\mathsf{Child} \sqsubseteq \mathsf{Human} & & \mathsf{Child} \sqcap \exists \mathsf{has\text{-}child}.\top \sqsubseteq \bot
\end{array} \tag{2.10}$$

This implies disjointness for concepts Father and Mother, as well as for concepts Grandmother and Father. If we add inclusion axiom ∃has-child.Human ⊑ Human saying that Humans can be children of only Humans, then, instead of stating *John* : Man in our ABox, we could equivalently write *John* : Male. In presence of the above axiom, this would imply the first assertion since *John* has a child *Bill* who was proven to be Human.

GCIs might be used not only for restricting existing concepts, but also for specifying *new* concepts *partially*. For example, in (2.10), we have partially defined a new concept Child by saying that those are Humans and they do not have their own children. (Of course, it is not appropriate to say that children are *exactly* those Humans that have no children). See [Brandt, 2004*a*] for more examples of using GCIs for conceptual modelling.

Baader et al. [2005] have demonstrated that reasoning in $\mathcal{EL}$ with GCIs and the empty concept, remains tractable. Moreover, an extension of $\mathcal{EL}$ with *simple role inclusion axioms* (or *role hierarchies*), remains polynomial as well [Brandt, 2004*a*; Baader et al., 2005]. A role inclusion axiom is an axiom of form $R \sqsubseteq S$, where R and S are role names. We also show that adding *conjunction of roles*, i.e., definitions of form $R \doteq S \sqcap H$, does not make reasoning in $\mathcal{EL}$ harder.

Table 2.11 CNF-translation for (extended) role hierarchies

TBox	$\mathcal{FO}$ – *translation*	**CNF** – *translation*	*clause type*
$A \doteq \bot$	$\forall x.[A(x) \leftrightarrow \bot]$	$\neg A(x)$	D9(A)
$R \sqsubseteq S$	$\forall xy.[R(x,y) \rightarrow S(x,y)]$	$\neg R(x,y) \vee S(x,y)$	DH1(R,S)
$R \sqcap S \sqsubseteq T$	$\forall xy.[((R(x,y) \wedge S(x,y)) \rightarrow T(x,y)]$	$\neg R(x,y) \vee \neg S(x,y) \vee T(x,y)$	DHX1(R,S,T)

D9 $\neg \underline{\boldsymbol{A}(x)}$;	DHX1 $\neg \underline{\boldsymbol{R}(\boldsymbol{a},\boldsymbol{b})} \vee \boldsymbol{S}(\boldsymbol{a},\boldsymbol{b})$;
DH1 $\neg \underline{\boldsymbol{R}(x,y)} \vee \boldsymbol{S}(x,y)$;	DHX2 $\neg \underline{\boldsymbol{S}(x,y)} \vee \neg \boldsymbol{R}(x,y) \vee \boldsymbol{T}(x,y)$;
	DHX3 $\neg \underline{\boldsymbol{A}(x)} \vee \neg \underline{\boldsymbol{S}(x,\boldsymbol{f}_{\boldsymbol{A}}(x))} \vee \boldsymbol{T}(x,\boldsymbol{f}_{\boldsymbol{A}}(x))$;

In order to extend our resolution decision procedure for $\mathcal{EL}$ described in Table 2.6, it suffices to enumerate all inferences between clause types from Table 2.5 and *new* clause types obtained in Table 2.11. These inferences are listed in Table 2.12.

Table 2.12 Summary of inferences for (extended) role hierarchies in $\mathcal{EL}$

A9.	$\mathtt{OR}$[C1; D9]	: $\underline{A(a)};\ \neg\underline{A(x)}\ \vdash\ \square$	: $\bot$
T10.	$\mathtt{OR}$[C3; D9]	: $\underline{A(x)};\ \neg\underline{A(x)} \vdash\ \square$	: $\bot$
T11.	$\mathtt{OR}$[C4; D9]	: $\neg A(x) \vee \underline{B(f_A(x))};\ \neg\underline{B(x)}\ \vdash\ \neg A(x)$	: D9
AH1.	$\mathtt{OR}$[C2; DH1]	: $\underline{R(a,b)};\ \neg\underline{R(x,y)} \vee S(x,y)\ \vdash\ S(a,b)$	: C2
TH1.	$\mathtt{OR}$[C5; DH1]	: $\neg A(x) \vee \underline{R(x,f_A(x))};\ \neg\underline{R(x,y)} \vee S(x,y)\ \vdash\ \neg A(x) \vee S(x,f_A(x))$	: C5
AHX1.	$\mathtt{OR}$[C2; DHX1]:	$\underline{R(a,b)};\ \neg\underline{R(a,b)} \vee S(a,b)\ \vdash\ S(a,b)$	: C2
AHX2.	$\mathtt{OR}$[C2; DHX2]:	$\underline{R(a,b)};\ \neg\underline{R(x,y)} \vee \neg S(x,y) \vee T(x,y)\ \vdash\ \neg S(a,b) \vee T(a,b)$	: DHX1
THX1.	$\mathtt{OR}$[C5; DHX2]:	$\neg A(x) \vee \underline{R(x,f_A(x))};\ \neg\underline{R(x,y)} \vee \neg S(x,y) \vee T(x,y)\ \vdash$ $\vdash\ \neg A(x) \vee \neg R(x,f_A(x)) \vee T(x,f_A(x))$	: DHX3
THX2.	$\mathtt{OR}$[C5; DHX3]:	$\neg A(x) \vee \underline{R(x,f_A(x))};\ \neg A(x) \vee \neg\underline{R(x,f_A(x))} \vee S(x,f_A(x)) \vdash$ $\vdash\ \neg A(x) \vee \neg A(x) \vee S(x,f_A(x))\ \vdash\ \neg A(x) \vee S(x,f_A(x))$	: C5

So, let us now compute a datalog program for reasoning in the above mentioned extensions of $\mathcal{EL}$. W.l.o.g. we may assume that (***i***) the empty concept occurs only in TBox-definitions of form $A \doteq \bot$, where A is a *concept name*, and (***ii***) role inclusion axioms are either of two forms: $R \sqsubseteq S$ or $R \sqcap S \sqsubseteq T$, where R, S and T are *role names*. In Table 2.11 we have listed **CNF**-translations for these new axioms and new clause types that result from it.[7] Encoding them as datalog rules, similarly as has been done for $\mathcal{EL}$, we obtain an extension of a datalog program given in Table 2.13, using which reasoning in $\mathcal{EL}$ with empty concept and extended role hierarchies can be performed.

Complexity bounds for the extension of our datalog program with the rules from Table 2.13, can be extracted using a similar analysis as has been performed in subsection 2.3.3. It is easy to see that the deductive closure of the initial database, can contain at most linear number of underlined atoms and at most quadratic number of dashed-underlined atoms. Hence a cubic bound on the number of prefix firings can be extracted for all rules given in Table 2.13 except for rule AHX2. For rule AHX2 we have a linear number of prefix firings for atom DHX2(R,S,T), however, for every choice of such prefix firing, there are two free variables left in the first atom: a and b. Hopefully, there are only linear number of pairs (a,b) which may occur in atoms of form C2(R,a,b), since, as can be easily seen, those must occur together in some initial atom. To summarise, reasoning in $\mathcal{EL}$ augmented with the empty concept and extended role hierarchies, can be performed in *cubic time* in the size of TBox

[7] We add postfix H to indicate that a clause originates from a simple role inclusion axiom (role **h**ierarchy), and HX to indicate that a clause originates from an e**x**tended inclusion axiom

Table 2.13 Additional datalog rules for reasoning with role hierarchies in $\mathcal{EL}$

A9.	$\bot$	$\leftarrow \mathsf{C1}(A, a),\ \mathsf{D9}(A);$
T10.	$\bot$	$\leftarrow \mathsf{C3}(A),\ \mathsf{D9}(A);$
T11.	$\mathsf{D9}(A)$	$\leftarrow \mathsf{C4}(A, B, f),\ \mathsf{D9}(B);$
AH1.	$\mathsf{C2}(S, \underline{a, b})$	$\leftarrow \mathsf{C2}(\boldsymbol{R}, \underline{a, b}),\ \underline{\mathsf{DH1}(\boldsymbol{R}, S)};$
TH1.	$\mathsf{C5}(A, S, f_A)$	$\leftarrow \mathsf{C5}(A, \boldsymbol{R}, f_A),\ \underline{\mathsf{DH1}(\boldsymbol{R}, S)};$
AHX1.	$\mathsf{C2}(S, \underline{a, b})$	$\leftarrow \mathsf{C2}(\boldsymbol{R}, \underline{\boldsymbol{a}, \boldsymbol{b}}),\ \dashuline{\mathsf{DHX1}(\boldsymbol{R}, S, \boldsymbol{a}, \boldsymbol{b})};$
AHX2.	$\mathsf{DHX1}(S, T, a, b)$	$\leftarrow \mathsf{C2}(\boldsymbol{R}, \underline{a, b}),\ \underline{\mathsf{DHX2}(\boldsymbol{R}, S, T)};$
THX1.	$\mathsf{DHX3}(A, R, T, f_A)$	$\leftarrow \mathsf{C5}(A, \boldsymbol{R}, f_A),\ \underline{\mathsf{DHX2}(\boldsymbol{R}, S, T)};$
THX2.	$\mathsf{C5}(A, S, f_A)$	$\leftarrow \mathsf{C5}(\boldsymbol{A}, \boldsymbol{R}, f_A),\ \dashuline{\mathsf{DHX3}(\boldsymbol{A}, \boldsymbol{R}, S, f_A)};$

and ABox.

2.4.2 Cross-Products of Concepts

Let us try to understand and explain *why* reasoning in $\mathcal{EL}$ and its mentioned extensions is tractable, whereas it is not for most other description logics. If we look carefully at the result of **CNF** transformations for $\mathcal{EL}$-TBox-es, we observe that we obtain only *Horn clauses*, i.e., clauses with at most one positive literal. However *Horn logic*, has the same expressive power as logic programming, thus is *undecidable* in general (see e.g., [Dantsin et al., 2001]). Hence, this fact does not even explain *decidability* for $\mathcal{EL}$ and its extensions. However, satisfiability for Horn clauses *without functional symbols* is decidable and even polynomial when the number of different variables in clauses is bounded. We can see that extensions described in the previous section result in clauses of this form. So, it might be reasonable to search for tractable extensions of $\mathcal{EL}$ among those constructors, which correspond to Horn clauses without functional symbols.

In literature on description logics [e.g., in Borgida, 1996], we find a role forming operator called a *cross-product of concepts*. This constructor allows one to form a *role expression* $C_1 \times C_2$ from two concepts C_1 and C_2, which intuitively corresponds to the Cartesian product of the sets correspondent to C_1 and C_2. For example, in our terminology of human relations we can write an axiom:

$$\mathsf{Grandmother} \times \mathsf{Child} \sqsubseteq \mathsf{likes} \tag{2.11}$$

expressing that every Grandmother likes every Child. Now, if we look at the **CNF**-translation for such axioms given in Table 2.14, we notice that they correspond to Horn clauses without functional symbols (again, we may assume that only concept names and role names are used in definitions). It is possible to show that resolution

Table 2.14 CNF-translation for cross-products of concepts

TBox	$\mathcal{FO}$ – *translation*	**CNF** – *translation*	*clause type*
$R \sqsubseteq A \times B$	$\forall xy.[R(x,y) \to (A(x) \wedge B(y))]$	$\neg R(x,y) \vee A(x)$ $\neg R(x,y) \vee B(y)$	$\mathsf{DP7}(R,A)$ $\mathsf{DP8}(R,B)$
$R \sqsupseteq A \times B$	$\forall xy.[(A(x) \wedge B(y)) \to R(x,y)]$	$\neg A(x) \vee \neg B(y) \vee R(x,y)$	$\mathsf{DP10}(A,B,C)$

for $\mathcal{EL}$ and these new clauses produces only finitely many clauses. Please find in Appendix A.4 the full list of inferences (there are quite many of those) and the resulted datalog program.

The extension of $\mathcal{EL}$ with cross-products, remains tractable even if the empty concept and simple role hierarchies are allowed. However, it becomes PSPACE-hard for *conjunction of roles*. The reason is that using extended role hierarchies, it is possible to express inclusion axioms with *universal* value restrictions of form $C \sqsubseteq \forall R.D$, or equivalently, inclusion axioms with *inverse roles* of form $\exists R^{-}.C \sqsubseteq D$, which were shown to cause intractability in [Baader et al., 2005]. Indeed, these axioms are expressible using three inclusion axioms: $C \times \top \sqsubseteq S$, $S \sqcap R \sqsubseteq H$ and $H \sqsubseteq \top \times D$, where S and H are fresh role names. However, we conjecture that $\mathcal{EL}$ with conjunction of roles remains polynomial if one admits cross-products only in axioms of form $C_1 \times C_2 \sqsubseteq R$.

2.4.3 Nominals

Nominal is a constructor that allows one to form a concept $\{a\}$ from an individual a. Intuitively, concept $\{a\}$ corresponds to a single-element set consisting of a. By combining nominals and the empty concept in $\mathcal{EL}$, one can express a so-called *unique name assumption* for individuals, namely that different individuals must correspond to different elements. For example, we can express that *John* and *Bill* are different persons using the first axiom below:

$$\begin{array}{c} \{\mathit{John}\} \sqcap \{\mathit{Bill}\} \sqsubseteq \bot \\ \hdashline \{\mathit{John}\} \times \{\mathit{Bill}\} \sqsubseteq \mathsf{has\text{-}child} \\ \{\mathit{John}\} \times \mathsf{Child} \sqsubseteq \mathsf{likes} \end{array} \tag{2.12}$$

Using cross-products of roles considered in the previous section, it is possible to express and generalise role assertions: the second axiom above is equivalent to a role assertion $(\mathit{John}, \mathit{Bill}) : \mathsf{has\text{-}child}$, which can be also equivalently written as $\{\mathit{John}\} \sqsubseteq \exists \mathsf{has\text{-}child}.\{\mathit{Bill}\}$, and the last axiom expresses that *John* likes all Children.

Nominal is another example of a constructor which corresponds to Horn clauses without functional symbols (see Table 2.15), but this time with *equality*.

Table 2.15 CNF-translation for nominals

TBox	$\mathcal{FO}$ – *translation*	**CNF** – *translation*	*clause type*
$\{a\} \sqsubseteq A$	$A(a)$	$A(a)$	$\mathsf{C1}(A,a)$
$\{a\} \sqsupseteq A$	$\forall x.(A(x) \rightarrow (x \simeq a))$	$\neg A(x) \vee x \simeq a$	$\mathsf{DO2}(A,a)$

Recently Baader et al. [2005] have proved that an extension of $\mathcal{EL}$ with nominals also remains polynomial. In this section we confirm and generalise this result using the *ordered paramodulation calculus*, which is a general-purpose calculus for the first-order logic with *equality*.

The *ordered paramodulation calculus* is an extension of the ordered resolution calculus with two inference rules given in Figure 2.2. The first rule called Ordered Paramodulation allows one to use an eligible equational atom $s \simeq t$ to perform replacement of a subterm in eligible literals of other clauses. For example, using this rule one can perform the following inference:

Figure 2.2 The ordered paramodulation calculus with selection

Ordered Paramodulation

$$\texttt{OP}: \frac{C \vee \underline{\boldsymbol{s}} \simeq \boldsymbol{t}^{\star} \quad D \vee \boldsymbol{L}[\underline{\boldsymbol{s}'}]}{C\sigma \vee D\sigma \vee L[t]\sigma}$$

[*where (i)* $\sigma = mgu(s,s')$; *(ii)* $s \simeq t$ *is eligible in clause* $C \vee s \simeq t$; *(iii)* $L[s']$ *is eligible in clause* $D \vee L[s']$; *(iv)* $t\sigma \not\succeq s\sigma$ *and (v)* s' *is not a variable.*]

Reflexivity Resolution

$$\texttt{RR}: \frac{C \vee \underline{\boldsymbol{s}} \not\simeq \underline{\boldsymbol{s}'}}{C\sigma}$$

[*where (i)* $\sigma = mgu(s,s')$ *and (ii)* $s \not\simeq s'$ *is eligible in clause* $C \vee s \not\simeq s'$.]

$$\neg A(x) \vee \underline{f_A(x)} \simeq a,\ \neg A(x) \vee R(\underline{f_A(x)}, b)\ \vdash\ \neg A(x) \vee \neg A(x) \vee R(a,b) \qquad (2.13)$$

Two important restrictions of the Ordered Paramodulation rule are (*iv*), saying that the replaced term should not be smaller than the term on which it is replaced and (*v*), saying that variables should not be replaced at all. The Reflexivity Resolution rule allows one to eliminate inequalities from clauses by unifying the respective terms.

The *ordered paramodulation calculus* is a sound and complete calculus for the first-order logic *with equality*. Consequently, we can use it to obtain sound and complete procedures for everything which can be expressed in this logic.

In Appendix A.5 we have applied the inference rules from Figure 2.1 and Figure 2.2 to clauses for the extensions of $\mathcal{EL}$ considered before, and *new* clauses for nominals given in Table 2.15, and demonstrated that only finitely many clauses could be produced. Our analysis proves that: (**1**) ordered paramodulation is a decision procedure for reasoning problems in the considered extensions of $\mathcal{EL}$ with nominals and (**2**) these reasoning problems are solvable in polynomial time in the size of ABox and TBox.

2.5 DL $\mathcal{EL}$ and Restricted Role-Value Maps

In the previous sections we have demonstrated how reasoning algorithms for several extensions of $\mathcal{EL}$ can be uniformly constructed step-by-step, by adding clause types for new constructors, and then closing the resulted set of clause types under inferences of an appropriate general-purpose calculus. There is, however, one more extension of $\mathcal{EL}$ considered by Baader [2003], for which, as it turns out, more sophisticated techniques are required to obtain even *decidability* results. This is an extension of $\mathcal{EL}$ with *restricted role-value maps*.

A (global) *role-value map* (short RVM) is an axiom of form $R_1 \circ \ldots \circ R_n \sqsubseteq S_1 \circ \ldots \circ S_m$, where $n \geq 1$, $m \geq 1$ and $\circ$ is a *composition operator* for roles. Axioms involving the *composition operator* will be often called *compositional axioms*. Such inclusion axiom expresses that for every sequence of elements a_0,.., a_n such that $(a_{i-1}, a_i) \in R_i$ for every $i = 1$,.., n, there exist elements b_0,.., b_m such that $b_0 = a_0$, $b_m = a_n$ and $(b_{j-1}, b_j) \in S_j$ for every $j = 1$,.., m.

Description logics with role-value maps have been considered in the context of the Kl-One system [Brachman & Schmolze, 1985]. Later it was realized in [Schmidt-Schauß, 1989], that such axioms make reasoning in description logics undecidable [see a related discussion in Donini, 2003], and $\mathcal{EL}$ is not an exception in this respect [Baader, 2003]. More precisely, Baader [2003] has proved that: (***i***) concept subsumption problem for $\mathcal{EL}$ augmented with role-value maps is *undecidable*, however (***ii***) for *restricted role-value maps* of form $R_1 \circ \ldots \circ R_n \sqsubseteq S$, $n \geq 1$, subsumption of concepts can be decided in polynomial time. Recently in [Baader et al., 2005], this result has been generalised to other extensions of $\mathcal{EL}$.

In this section we demonstrate how decision procedures for reasoning in $\mathcal{EL}$ with restricted role-value maps can be obtained using saturation-based calculi.

2.5.1 Undecidability for Some Extensions of $\mathcal{EL}$ with Role-Value Maps

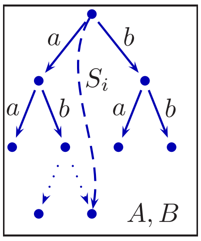
Before we describe a resolution strategy for an extension of $\mathcal{EL}$ with simple role-value maps, let us review some further possible extensions of this description logic. It has been demonstrated in [Brandt, 2004*a*; Baader et al., 2005], that reasoning in $\mathcal{EL}$ becomes intractable as long as some additional constructors are added, in particular, disjunction, universal value restrictions or inverse roles. It turns out, that in presence of restricted role-value maps, this distinction is even more dramatical. Not only tractability of $\mathcal{EL}$ with restricted RVMs is lost when these constructors are added, but actually, the resulted description logics become *undecidable*.

For proving undecidability for extensions of $\mathcal{EL}$ with restricted RVMs, we use a reduction from a well-known unsolvable problem about the intersection emptiness of context-free languages. A *context-free* grammar in Chomsky normal form is defined by a set of *production rules* of forms (**1**) $A \to BC$ or (**2**) $A \to a$, where capital letters A, B and C are called *non-terminals*, or *variables* and small letters a, b, etc., are called *terminals*. A *language generated by* a grammar from a *start symbol* S, is the set of all strings consisting of terminal symbols only, that are obtained by rewriting S using the production rules. The following problem is undecidable [see Hopcroft & Ullman, 1979]: *Given two context-free grammars in Chomsky normal form, with disjoint sets of non-terminals, start symbols S_1 and S_2, respectively, and common terminal symbols a and b, check, if languages generated from S_1 and S_2 intersect.*

Our reduction is close in spirit to some other undecidability proofs for logics with compositional axioms [*see* e.g., Baldoni, Giordano & Martelli, 1998; Ganzinger et al., 1999]. Given two grammars defined by sets of production rules, we introduce new role names R_A and R_a for every non-terminal A and terminal a. For every production rule $A \to BC$ and $A \to a$, we form inclusion axioms $R_B \circ R_C \sqsubseteq R_A$ and $R_a \sqsubseteq R_A$ respectively. Finally, we add the following definition to TBox:

$$A \doteq \exists R_a.A \sqcap \exists R_b.A \sqcap B \tag{2.14}$$

where A, B are fresh concept names and a, b are the non-terminals of our grammars.



Definition (2.14) enforces a tree-like structure illustrated in the figure to the right: every element of concept A must be an element of concept B and must be connected with other two elements of concept A via roles R_a and R_b respectively. Now, if a word $\overline{w} = w_1w_2...w_k$ over letters $w_j \in \{a, b\}$, $1 \leq j \leq k$, belongs to a language generated by S_i, $i = 1, 2$, i.e., in symbols: $S_i \to w_1w_2...w_k$, then we must have $R_{w_1} \circ \ldots \circ R_{w_k} \sqsubseteq R_{S_i}$ according to our compositional axioms. This means that for every element x from A there exists an element y from B, connected with a chain of roles labelled by letters from $\overline{w}$, such that $(x, y) \in R_{S_i}$. Hence languages generated by our grammars have a common word w if and only if the following property holds:

> *In every model of* (2.14), *every element from A is connected to some element from B via both roles R_{S_1} and R_{S_2}.* (2.15)

To complete our reduction and thereby prove undecidability, it suffices to express the last property using additional constructors:

Conjunction of roles. First of all, property (2.15) can be easily expressed using conjunction of roles. Indeed, subsumption ?- $A \sqsubseteq \exists(R_{S_1} \sqcap R_{S_2}).B$ w.r.t. TBox (2.14) holds if and only if (2.15) holds.

Disjunction of concepts. Now we show how to express property (2.15) using disjunction of concepts. Let us add the following definition to TBox (2.14):

$$B \doteq C \sqcup D \tag{2.16}$$

where C and D are fresh concept names. We claim that subsumption ?- $A \sqsubseteq (\exists R_{S_1}.C \sqcup \exists R_{S_2}.D)$ w.r.t. the resulted TBox holds if and only if property (2.15) holds. Indeed, as we have demonstrated, (2.15) implies subsumption $A \sqsubseteq \exists(R_{S_1} \sqcap R_{S_2}).B$, which together with (2.16) yields our subsumption as follows:

$$\begin{aligned} A \;\sqsubseteq\; \exists(R_{S_1} \sqcap R_{S_2}).B \;=\; \exists(R_{S_1} \sqcap R_{S_2}).(C \sqcup D) \;=\; \\ =\; \exists(R_{S_1} \sqcap R_{S_2}).C \sqcup \exists(R_{S_1} \sqcap R_{S_2}).D \;\sqsubseteq\; \exists R_{S_1}.C \sqcup \exists R_{S_2}.D. \end{aligned}$$

Conversely, suppose that property (2.15) does not hold, i.e., there exists a model $\mathcal{M}$ and an element $x \in A$ such that for every $y \in B$ we have $(x, y) \notin (R_{S_1} \sqcap R_{S_2})$. Then we can extend this model by defining concepts C and D as follows: (***i***) $y \in C$ *iff* $y \in B$ and $(x, y) \notin R_{S_1}$; (***ii***) $y \in D$ *iff* $y \in B$ and $(x, y) \notin R_{S_2}$. It is easy to see that these definitions satisfy (2.16), but our subsumption does not hold: $x \in A$, but $x \notin \exists R_{S_1}.C$ and $x \notin \exists R_{S_2}.D$.

Universal value restrictions. For universal value restrictions, we consider the following subsumption problem: ?- $(A \sqcap \forall R_{S_2}.C) \sqsubseteq \exists R_{S_1}.C$ w.r.t. TBox (2.14). Again, (2.15) implies this subsumption:

$$A \sqcap \forall R_{S_2}.C \;\sqsubseteq\; \exists(R_{S_1} \sqcap R_{S_2}).B \sqcap \forall R_{S_2}.C \;\sqsubseteq\; \exists(R_{S_1} \sqcap R_{S_2}).(B \sqcap C) \;\sqsubseteq\; \exists R_{S_1}.C.$$

If (2.15) does not hold for some model $\mathcal{M}$, then, in particular, it does not hold for a restriction of $\mathcal{M}$ to elements of A. By defining x and C like in the case with disjunction, we can notice that for every y with $(x, y) \in R_{S_2}$, we have $y \in A \sqsubseteq B$. Since we must also have $(x, y) \notin R_{S_1}$, by (i), we have $y \in C$, and so $x \in (A \sqcap \forall R_{S_2}.C)$. However, $x \notin \exists R_{S_1}.C$, since there is no $y \in C$ such that $(x, y) \in R_{S_1}$ because of (i). Hence our subsumption relation does not hold.

Inverse roles. Finally, we make a similar construction for inverse roles using the subsumption query: ?- $(A \sqcap C) \sqsubseteq (\exists R_{S_1}.\exists R^-_{S_2}.C)$. This subsumption can be derived from (2.15) as follows:

$$\begin{aligned} A \sqcap C \;\sqsubseteq\; A \sqcap \forall R_{S_2}.\exists R^-_{S_2}.C \;\sqsubseteq\; \exists(R_{S_1} \sqcap R_{S_2}).B \sqcap \forall R_{S_2}.\exists R^-_{S_2}.C \;\sqsubseteq \\ \sqsubseteq\; \exists(R_{S_1} \sqcap R_{S_2}).(B \sqcap \exists R^-_{S_2}.C) \;\sqsubseteq\; \exists R_{S_1}.\exists R^-_{S_2}.C. \end{aligned}$$

Conversely, if x is defined like before, then we can take $C := \{x\}$, and so $x \in (A \sqcap C)$, but $x \notin \exists R_{S_1}.\exists R^-_{S_2}.C$, since there is no y such that $(x, y) \in R_{S_1}$, and $(y, x) \in R^-_{S_2}$.

The above reductions prove that every extension of $\mathcal{EL}$ (without GCIs) with restricted RVMs and either *conjunction of roles*, or *disjunction of concepts*, or *universal value restrictions*, or *inverse roles* is undecidable.

2.5.2 A Resolution Strategy for $\mathcal{EL}$ with Restricted Role-Value Maps

As we have seen in the previous section, the extension of $\mathcal{EL}$ with restricted role-value maps is very fragile w.r.t. decidability, as most of its extensions appear to be undecidable. This means *every* resolution strategy for this description logic will simply not work when applied for its extensions. This is a rather strong indicator for difficulties in designing of such a resolution strategy, since it must take into consideration that our logic does *not* contain certain constructors. And indeed, difficulties with composition axioms do not hesitate to show up.

Difficulties with compositional axioms

Consider a *transitivity axiom* $T \circ T \sqsubseteq T$ for a role T, which is an instance of restricted role-value maps. This axiom corresponds to clause of form:

$$\mathsf{T}.\quad \neg\underline{T(x,y)} \;\vee\; \neg\underset{\cdots}{T(y,z)} \;\vee\; T(x,z) \qquad \textit{(transitivity)}$$

Although this is a Horn clause without functional symbols, we will see that it does not behave "well" under ordered resolution. Since the positive literal of this clause is not maximal, we have two choices for selecting its negative literal. We show that both of these choices result in infinite number of clauses derived.

Suppose we have an additional clause resulted from axiom $A \sqsubseteq \exists T.\top$:

$\mathsf{C}.1\ \neg A(x) \vee \underline{T(x, f_A(x))}$

If the first literal in clause T is selected, we obtain an infinite number of inferences with the transitivity clause:

$\mathtt{OR}[\mathsf{C}.1;\mathsf{T}]:\ \neg A(x) \vee \underline{T(x,f_A(x))};\ \neg\underline{T(x,y)} \vee \neg T(y,z) \vee T(x,z)\ \vdash$

$\vdash\ \mathsf{D}.1\ \neg A(x) \vee \neg\underline{T(f_A(x),z)} \vee T(x,z)$

$\mathtt{OR}[\mathsf{C}.1;\mathsf{D}.1]:\ \neg A(x) \vee \underline{T(x,f_A(x))};\ \neg A(x) \vee \neg\underline{T(f_A(x),z)} \vee T(x,z)\ \vdash$

$\vdash\ \mathsf{C}.2\ \neg A(f_A(x)) \vee \neg A(x) \vee \underline{T(x,f_A(f_A(x)))}$

$\mathtt{OR}[\mathsf{C}.2;\mathsf{T}]:\ \neg A(f_A(x)) \vee \neg A(x) \vee \underline{T(x,f_A(f_A(x)))};\ \neg\underline{T(x,y)} \vee \neg T(y,z) \vee T(x,z)\ \vdash$

$\vdash\ \mathsf{D}.2\ \neg A(f_A(x)) \vee \neg A(x) \vee \neg\underline{T(f_A(f_A(x)),z)} \vee T(x,z)$

$\mathtt{OR}[\mathsf{C}.2;\mathsf{D}.2]:\ \neg A(f_A(x)) \vee \neg A(x) \vee \underline{T(x,f_A(f_A(x)))};$

$\neg A(f_A(x)) \vee \neg A(x) \vee \neg\underline{T(f_A(f_A(x)),z)} \vee T(x,z)\ \vdash$

Continued on next page

$\vdash\ \neg A(f_A(x)) \vee \neg A(x) \vee \neg A(f_A(x)) \vee \neg A(x) \vee T(x, f_A(f_A(f_A(f_A(x)))))\ \vdash$

$\vdash\ \mathsf{C.3}\ \neg A(f_A(x)) \vee \neg A(x) \vee \underline{T(x, f_A(f_A(f_A(f_A(x)))))}$

.... etc.

Similarly, selection of the second negative literal in T yields infinite number of clauses:

$\mathtt{OR}[\mathsf{C.1}; \mathsf{T}] : \neg A(x) \vee \underline{T(x, f_A(x))};\ \neg T(x, y) \vee \neg \underline{T(y, z)} \vee T(x, z)\ \vdash$

$\vdash\ \mathsf{C.2}\ \neg T(x, y) \vee \neg A(y) \vee \underline{T(x, f_A(y))}$

$\mathtt{OR}[\mathsf{C.2}; \mathsf{T}] : \neg T(x, y) \vee \neg A(y) \vee \underline{T(x, f_A(y))};\ \neg T(x, y) \vee \neg \underline{T(y, z)} \vee T(x, z)\ \vdash$

$\vdash\ \mathsf{C.3}\ \neg T(x_1, x) \vee \neg T(x, y) \vee \neg A(z) \vee \underline{T(x_1, f_A(y))}$

.... etc.

As we see, both strategies result in either increase in the depth or in the length of the produced clauses, which means that the saturation process does not terminate. It seems like nothing can be done to fix this behaviour. However, imagine for a moment that we can select the *positive* literal of the transitivity clause:

$$\mathsf{T.}\quad \neg T(x, y)\ \vee\ \neg T(y, z)\ \vee\ \underline{T(x, z)} \qquad \textit{(transitivity)}$$

In this case, none of the above inferences is possible. But there might be other inferences with clauses containing T negatively, say, with clauses resulted from translation of an inclusion axiom of form $\exists T.B \sqsubseteq C$ (see Table 2.3):

$\mathsf{D.1}\ \neg \underline{T(x, y)} \vee \neg B(y) \vee C(x)$

Resolution between the transitivity clause and this clause results in inference:

$\mathtt{OR}[\mathsf{T}; \mathsf{D.1}] : \neg T(x, y) \vee \neg T(y, z) \vee \underline{T(x, z)};\ \neg \underline{T(x, y)} \vee \neg B(y) \vee C(x)\ \vdash$

$\vdash\ \mathsf{D.2}\ \neg T(x, y) \vee \neg T(y, z) \vee \neg B(z) \vee C(x)$

It looks like we have the same problem again, since now we have to select some of the negative literals in clause D.2, which then can be resolved with the transitivity clause producing similar, but longer clauses. Now the second trick which helps to avoid this problem is to *split* clause D.2 into two *shorter* clauses. This can be formally done using an additional optional rule:

Splitting through New Predicate Symbol

$$\mathsf{SPP} : \frac{[\![C \vee D]\!]}{\begin{array}{l} C \vee \quad u_C(\overline{x}) \\ D \vee \neg u_C(\overline{x}) \end{array}} \tag{2.17}$$

$$\left[\begin{array}{l} \textit{where (i) } \overline{x} \textit{ are the shared variables of subclauses } C \textit{ and } D\textit{; and} \\ \textit{(ii) } u_C(\overline{x}) \textit{ is a fresh atom introduced for } C. \end{array}\right]$$

Rule (2.17) introduces a "definition" $u_C(\overline{x})$ for the negation of a subclause C, which is then used for splitting this clause into smaller ones. The splitting rule (2.17) was argued to be useful for speeding-up a saturation process in theorem provers [Riazanov & Voronkov, 2001; de Nivelle, 2001]. Here we demonstrate that using this rule, *termination* of a saturation process can be regained.

Let us see how the Splitting through New Predicate Symbol can be applied to clause D.2 above. This clause can be partitioned into two subclauses:

D.2 = $[\neg T(y,z) \vee \neg B(z)] \vee [\neg T(x,y) \vee C(x)]$, with a common variable y.

Now, using a fresh atom $T^B(y)$ for the first part of the clause, we can split D.2 into the following two clauses according to rule (2.17):

D.2.1 $\neg T(y,z) \vee \neg B(z) \vee T^B(y)$; D.2.2 $\neg T(x,y) \vee \neg T^B(y) \vee C(x)$.

Semantically, atom $T^B(y)$ represents the set of all elements from which some element from B is reachable via T, or, writing this in DL-syntax, $T^B \doteq \exists T.B$. Speaking about DL-syntax, note that clause D.2 expresses the inclusion axiom: $\exists T.\exists T.B \sqsubseteq C$, and the splitting rule essentially replaces it with two inclusion axioms: $\exists T.B \sqsubseteq T^B$ and $\exists T.T^B \sqsubseteq C$, that correspond to clauses D.2.1 and D.2.2 respectively.

The resolution strategy

Unfortunately, resolution with selection of positive literals is not complete for general clauses. However, for the *Horn clauses*, which we essentially obtain for extensions of $\mathcal{EL}$, selection of positive literals is admissible (see related discussion in [de Nivelle, 1995, Section 6.7] and in [Bachmair & Ganzinger, 2001, Section 7.2]). Nevertheless, we will *not* rely on this result to leave a possibility of extending $\mathcal{EL}$ with constructors that do not correspond to Horn clauses. Instead, we make use of *renaming techniques*[8] to *simulate* the effect of positive selection. The idea is to replace the

[8] Renaming techniques are often used for proving completeness of different resolution strategies, including *resolution with free selection* discussed here [see Bachmair & Ganzinger, 2001]

transitivity clause T with two clauses:

$$\mathsf{T}.1 \quad \neg T(x,y) \vee \neg T(y,z) \vee \underline{\neg T^{\neg}(x,z)}; \quad \text{and} \quad \mathsf{T}.2 \quad T^{\neg}(x,y) \vee \underline{T(x,y)},$$

where $T^{\neg}(x,y)$ is a new atom that corresponds to the negation of $T(x,y)$. Now we are allowed to select the last literal in clause $\mathsf{T}.1$. To make sure that the transitivity clause will not be derived back from $\mathsf{T}.1$ and $\mathsf{T}.2$, we make atom $T^{\neg}(x,y)$ slightly smaller in ordering than atom $T(x,y)$.

In the general case, we may assume that our terminologies contain only compositional axioms of form $S \circ T \sqsubseteq H$, since longer inclusion axioms can be split into shorter ones by introducing auxiliary role names. Such axioms are translated to clauses analogously according to Table 2.18. Please see Appendix A.6 to find the details of how our strategy works in this case. It is shown that resolution produces only clauses of finitely many types, and hence, gives a polynomial decision procedure for $\mathcal{EL}$ with restricted RVMs.

Table 2.18 CNF-translation for restricted role-value **m**aps

TBox	$\mathcal{FO}$ – *translation*	**CNF** – *translation*	*clause type*
$S \circ T \sqsubseteq H$	$\forall xyz.[(S(x,y) \wedge T(y,z)) \rightarrow$ $\rightarrow H(x,z)]$	$\neg S(x,y) \vee \neg T(y,z) \vee \neg H^{\neg}(x,z)$ $H^{\neg}(x,y) \vee H(x,y)$	$\mathsf{DM12}(S,T,H)$ $\mathsf{CM4}(H)$

2.6 First Results

The algorithms for checking subsumption in description logic $\mathcal{EL}$ and its extensions are quite intriguing because they have provably polynomial complexity in the worst case. This is not true for the standard tableau-based procedures: Brandt [2004*b*] gave an example showing that such procedures would require exponentially many steps in the worst case for checking concept subsumption in $\mathcal{EL}$. However these procedures are not really designed to be optimal in the worst case but rather *efficient for the average case*, or for so-called "real wold problems" – it is well-known that for many EXPTIME-complete description logics, tableau procedures might have NEXPTIME or even 2NEXPTIME behaviour in the worst case. Still, these procedures deliver good performance in practice because of a number of sophisticated optimisation techniques employed [Horrocks et al., 2000].

What we want to say, is that theoretical results might not be convincing enough in favour of specialised *completion-based*/*saturation-based* procedures for $\mathcal{EL}$ presented in this chapter, over the standard tableau-based procedures. In order to fully validate our approach we have performed a series of experiments, the results of which we are proud to present in this section.

Description of the test data

For our experiments, we have generated a collection of $\mathcal{EL}$ terminologies, consisting of simple concept definitions, which are possible for $\mathcal{EL}$ of two types:

$$(\mathbf{1})\ A_i \doteq A_j \sqcap A_k; \qquad \textit{and} \qquad (\mathbf{2})\ A_i \doteq \exists R_j.A_k; \tag{2.18}$$

where A_i are *concept names* and R_j are *role names*. These definitions were generated according to the following parameters:

- **C** - *the number of* ***c****oncept names used in TBox;*
- **R** - *the number of* ***r****ole names used in TBox;*
- **DP** - *probability that a concept name is defined (****d****efinition* ***p****robability);*
- **CP** - *probability that a definition is of type (1) from* (2.18) *(****c****onjunction* ***p****robability).*

Given a combination of these parameters, we generate a TBox by providing at most one definition for every concept name with probability **DP**. This definition is of type (1) with probability **CP** and otherwise is of type (2). Concept names and role names in the right hand side of these definitions are chosen randomly from **C**, respectively **R** candidates.

Intuitively, the large is the value for **DP**, the more concept dependences (subsumptions) there are is the resulted TBox. On the other hand, the value for **CP** corresponds to the ratio of propositional definitions in TBox. This should normally affect the length of tableau branches.

When we have decided on the possible values for parameters **C** and **R**, we took into account that the size of serious terminologies goes as far as up to several hundred thousand definitions (e.g., Galen [Rector, 2002], or Snomed [Spackman et al., 1997]). Since our implementation is only prototypical and relatively naïve, we have decided to test TBoxes with up to $10,000$ definitions, i.e., for $\mathsf{C} \leq 10,000$.

The number of role names used in terminologies is usually rather limited compared to the number of concept names: roles often correspond to fixed attributes or fields in records. Our conjecture was that classification becomes harder when there are less roles in TBox (and hence more concept dependences). So, we decided to perform tests only for $\mathsf{R} \leq 10$.

The ranges for probabilities that we have considered are: $0.5 \leq \mathsf{DP} \leq 1$ and $0 \leq \mathsf{CP} \leq 1$.

Description of systems and experiment conditions

We have implemented two *completion-based* procedures for classification of $\mathcal{EL}$ terminologies. The first one is based on completion rules (2.4) formulated in [Brandt, 2004*a*; Baader et al., 2005]. The second one is based on a datalog program given in Table 2.7 that we have derived by ordered resolution.

Both procedures have been implemented using XSB Prolog system [Sagonas et al., 1994], which has a usual top-down depth-first evaluation but it is additionally enhanced with so-called *tabling*. *Tabling*, also called *memoing* is a technique that helps to avoid possible loops in top-down evaluation of recursive logical programs by blocking subgoals that have been already processed. This makes the XSB system particularly useful for evaluation of queries in *datalog*, since it is guaranteed to terminate. The source codes for both programs can be found in Appendix A.7.

In our experiments we call the first program based on **c**ompletion **r**ules (2.4) by XSB-CR, and the second program from Table 2.7 based on **o**rdered **r**esolution, by XSB-OR. We have run both programs on randomly generated TBoxes and have measured the CPU-time spent on *full classification* of these terminologies. For comparison, we have also classified these TBoxes using RACER system. RACER [Haarslev & Möller, 2001] is a highly optimised tableau-based system for reasoning in TBoxes and ABoxes for very expressive description logics.

All tests were performed on machines equipped with 2.60GHz Intel Xeon processors with 2GB of memory running Linux. We have used XSB version 2.7.1 and RACER version 1.7.23. For all tests the CPU time-limit was set to 30 minutes (= 1800 seconds) and memory limit to 1GB. XSB programs where run with an option -S that enables *subsumption-based tabling*.

Results of the experiments

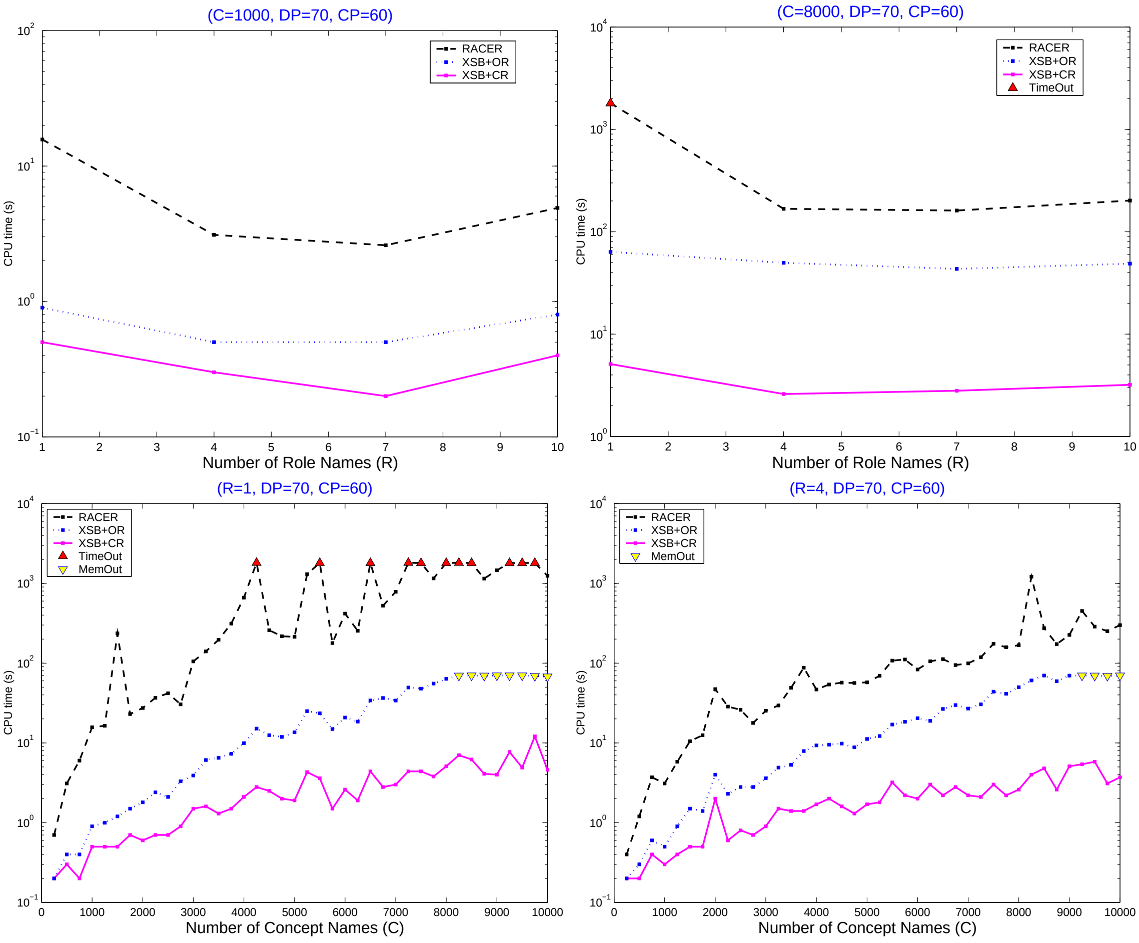
The results of experiments turned out to be very exciting. Not only completion-based procedures have outperformed RACER– they where actually $10 - 100$ times faster. Both XSB-programs have either computed classification within 70 seconds, or ran out of memory. RACER turned out to be more memory efficient: none of its runs have been killed by memory-out, but by time-out. However this did not actually resulted in that more TBoxes have been classified.

Regarding the comparison between completion-based procedures, XSB-CR has been generally faster than XSB-OR, for simple problems even in several times, however for harder problems their performance were very close. This was expected, since program (2.4) has only one rule that causes the cubic complexity (**CR4**), whereas program given in Table 2.7 has two such rules (T.5 and S.3).

Now, to the detailed analysis. In Figure 2.3 we have compared performance of our test systems with respect to the number of role names (**R**) used in generated TBoxes. Recall, that our conjecture was that classification becomes harder when there are only few role names. And indeed, for the RACER system we can observe this behaviour: when moving from one role name to four role names, its performance changes in an order of magnitude. Surprisingly, for completion-based systems the difference is almost negligible, especially for big problems. Another surprise is that with bigger values for **R**, timings are actually starting to grow.

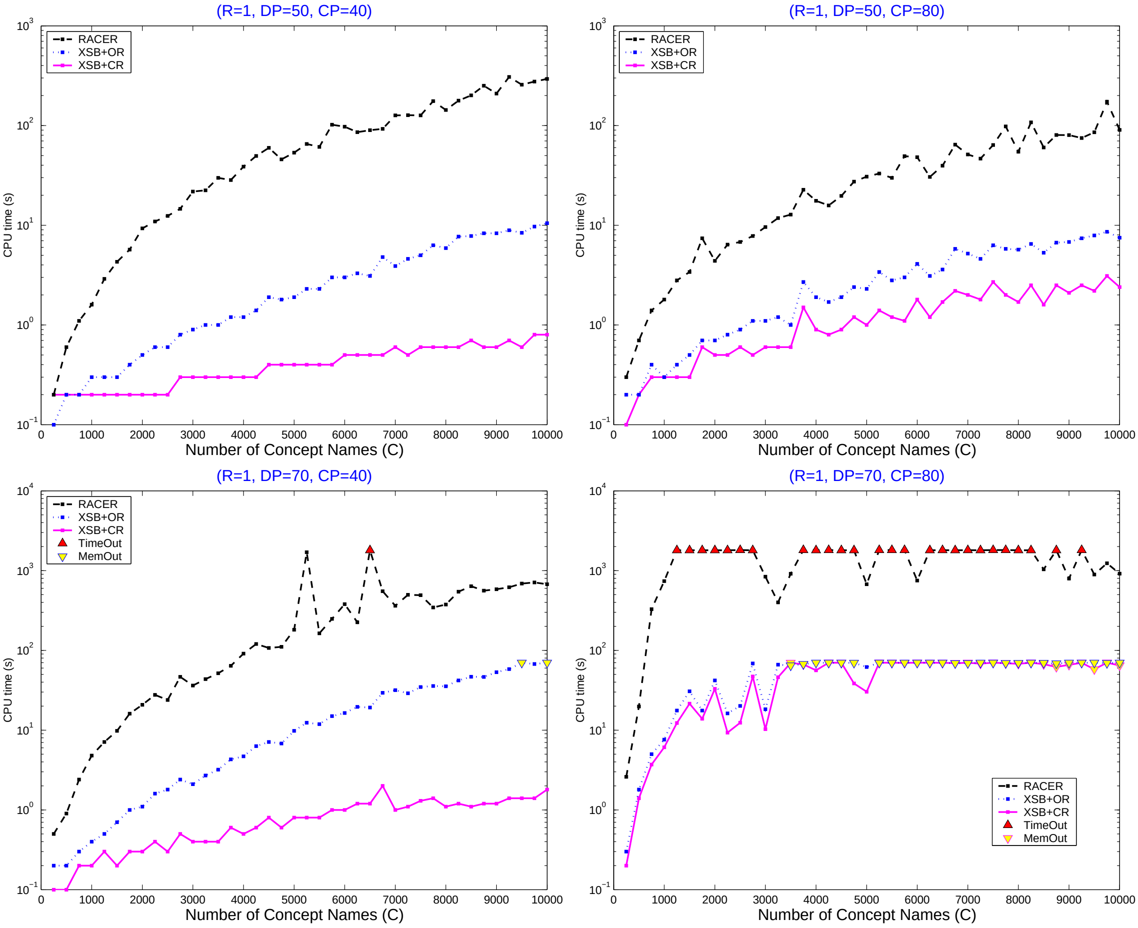
In the lower part of Figure 2.3, we performed a detailed comparison between our systems for two particular settings for the number of different roles: **R** $= 1$ in the left graph and **R** $= 4$ in the right graph, and setting the remaining parameters to their average values: **DP** $= 70$ and **CP** $= 60$. Again the difference for the completion-based procedures is minimal. RACER performs considerably better in the second case, where it was able to solve all problems.

Figure 2.3 Performance comparison w.r.t. the number of roles



In Figure 2.4 we made a detailed analysis for performance of our systems w.r.t. the last two parameter of our tests – the probability of definition and the ratio for the types of definitions. We made four cuts for the results of our tests: for **DP** = 50, **DP** = 70, and **CP** = 40, **CP** = 80. The behaviour of our systems w.r.t. to these parameters appeared to be not so determined as in the previous cases. We see that for "sparse" knowledge bases (with **DP** = 50), classification is easier for Racer with more conjunctions, but is harder for XSB-CR and there is almost no difference for XSB-OR. In contrast, for "dence" knowledge bases with bigger ratio of concept definitions (**DP** = 70) the difference w.r.t. the last parameter **CP** is drastic: when there are only three problems that where not solved by some system for **CP** = 40, with **CP** = 80 *most* of the problems have not been solved by *every* system, and this is already for a moderate number of concepts, starting from **C** = 1250. Note that

Figure 2.4 Detailed performance comparison w.r.t. the definition probability and conjunction probability

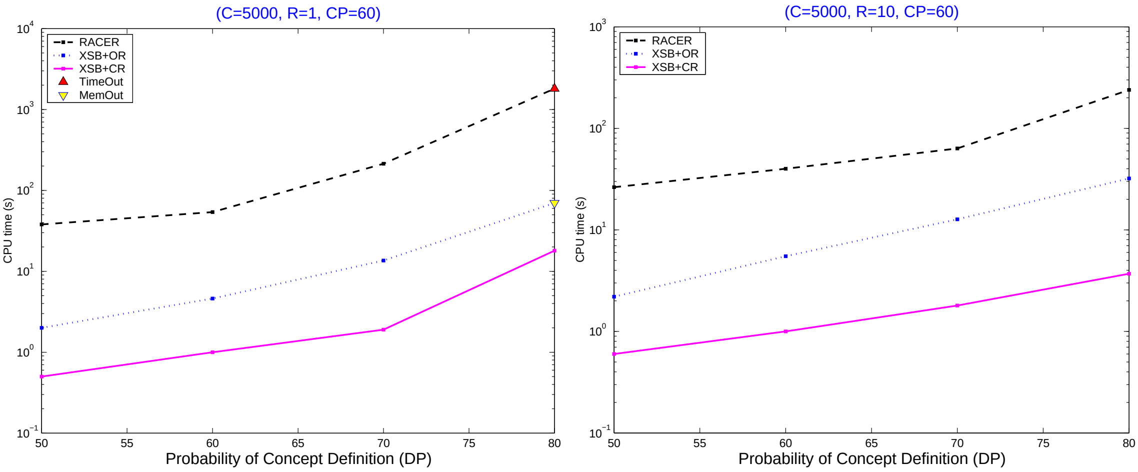


the timings for completion-based systems XSB-OR and XSB-CR become close with larger values of **CP**. This will be also confirmed in our last comparison set.

In Figure 2.5 we have compared performance of our systems w.r.t. the values for probability of concept definitions (**DP**). No surprises – performance for all systems becomes generally worse for more concept definitions, as was expected. What is strange is that all systems exhibit a definite exponential behaviour w.r.t. **DP** (the time scale in all graphs is logarithmic).

In Figure 2.6 we have compared our systems for TBoxes with moderate number of concept names **C** ≤ 3000, but for very difficult instances for other parameters: $80 \leq$ **DP** ≤ 100 and $50 \leq$ **CP** ≤ 100. In the first figure we see that for purely propositional TBoxes (with **CP** $= 100$) there is no essential difference in performance w.r.t. the number of concept definitions. However, when we fix this value to **DP** $:= 90$, we

Figure 2.5 Performance comparison w.r.t. the definition probability



notice that for all completion-based systems, the hardest TBox-es are not purely propositional, as could be conjectured from the previous analysis, but for **CP** = 80

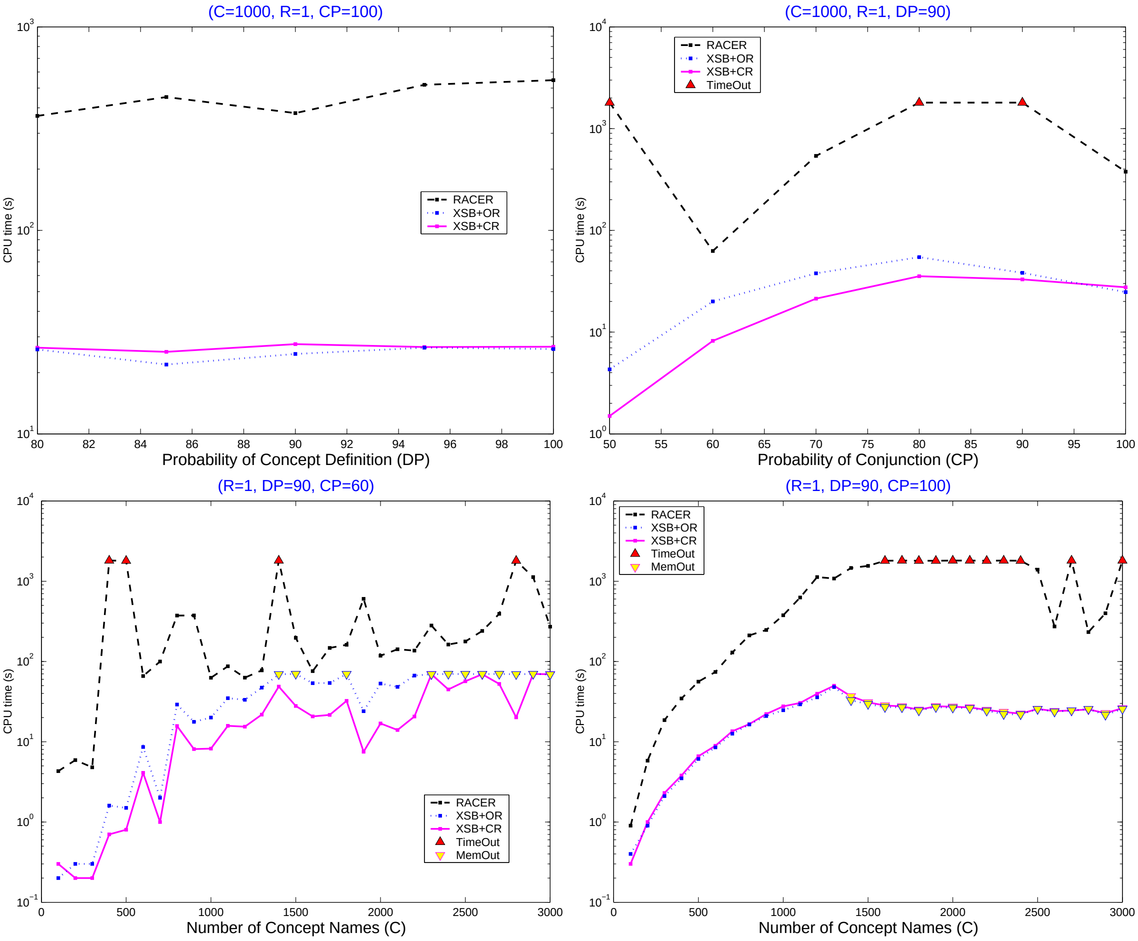
Finally, we compare general performance of our systems for very dense TBoxes (**DP** = 90). We see that for TBoxes with mixed types of definitions, the systems perform less stable than in the purely propositional case. The last sets of parameters appeared also to be the hardest for completion-based systems (which perform here almost identically), probably because they result in too many subsumption relations between concepts which simply do not fit into the memory.

Conclusions

Our experiments have demonstrated that all systems that we have compared, are suitable for classifying average-sized $\mathcal{EL}$-terminologies. The completion-based systems perform uniformly much faster than RACER, sometimes in a couple of orders of magnitudes, but consumes lots of memory. This is not very surprising, since RACER solves subsumption tests one-by-one by employing depth-first search, whereas completion-based systems classify whole TBox at once and keep all intermediate results. There is no absolute “winner” for our tests: all systems were not able to solve some problems which others did. Out of the total number of 4088 TBoxes that we have tested, RACER was *not* able to classify 710 TBoxes (17,4%), XSB-OR – 828 (20,3%) TBoxes, and XSB-CR – 538 (13,2%). Among completion-based systems, XSB-CR performed uniformly better than XSB-OR. In particular there were *no* TBoxes that XSB-OR has classified but XSB-CR not.

As a conclusion, completion-based algorithms have shown to be very promising for reasoning in $\mathcal{EL}$, compared to the traditional tableau-based algorithms. They

Figure 2.6 Performance comparison for very hard problems



provide an excellent time performance, but, have some issues with the memory consumption, which however might be resolved by applying some memory optimisation techniques (for example by storing only taxonomy, rather than all subsumption relations).

The comparison of systems on a small description logic $\mathcal{EL}$ might seem to be unfair, since RACER is optimised for reasoning in very expressive description logics. However, one should take into account that conjunction and existential restrictions are the mostly used constructors in ontologies, so it makes sense to develop optimisation techniques for these particular constructors within expressive description logics as well.

It would be also nice to have a comparison for real knowledge bases rather than for randomly generated ones. In the moment we do not have enough freely available candidates, although the TAMBIS ontology seems to be expressible in $\mathcal{EL}$ with simple role hierarchies, and is worth trying.

2.7 Conclusions

In this chapter we demonstrated how one can design algorithms for reasoning in description logics using saturation-based theorem proving. We derived a polynomial-time procedure for evaluation of subsumption and instance queries in a simple description logic $\mathcal{EL}$, and then extended this procedure for many other constructors. It turned out that our procedures can be implemented as datalog programs, which made it possible to use the standard deductive database systems for evaluation of DL-queries, and perform more detailed complexity analysis of the procedures. The main contributions of this chapter can be summarised as follows:

1. We derived *saturation-based* reasoning procedures for the following DLs: (**1**) $\mathcal{EL}$ + GCIs + *role hierarchies* (*cubic time*); (**2**) DL from (1) + *conjunctions of roles* (*cubic time*); (**3**) DL from (1) + *cross-products of concepts*; (**4**) DL from (2) and DL from (3) + *Nominals*; (**5**) $\mathcal{EL}$ + *restricted role-value maps* (*$O(n^5)$-time*).

2. We demonstrated *undecidability* for extensions of DL $\mathcal{EL}$ with *restricted role-value maps* and one of the following constructors: *conjunction of roles*, *disjunction of concepts*, *universal value restriction*, or *inverse roles*.

3. We implemented two procedures for $\mathcal{EL}$ (the one that we derived and the one from [Brandt, 2004*a*; Baader et al., 2005]) within XSB Prolog system, and demonstrated that they outperform the highly optimised tableau-based reasoner RACER.

2.8 Related Works

The idea of using general-purpose theorem provers for reasoning in description logics is probably as old as description logics themselves. Schild [1991] has observed a direct correspondence between the *basic description logic* $\mathcal{ALC}$ and a multi-modal version of modal logic $\mathbf{K}$. This opened possibilities for reasoning in description logics through the first-order logic, using relational and functional translations proposed, say, by Ohlbach [1991] for modal logics. Independently, Fermüller et al. [1993] following works of Joyner Jr. [1976], proposed several resolution-based strategies for fragments of first-order logic, including the fragment induced by relational translation of $\mathcal{ALC}$. These and other ideas have been implemented in system MSPASS

[Hustadt et al., 1999], which is an extension of the general-purpose theorem prover Spass with different translations for modal formulas. Although MSpass was primarily aimed at modal logics, it can be used for checking satisfiability of $\mathcal{ALC}$-concepts.

Resolution is not the only general theorem-proving method which has been tried for modal and description logics. Paramasivam & Plaisted [1998] argued that theorem provers based on search of *finite models* can be used for checking concept subsumption in virtually every description logic with a finite model property (and there are surprisingly many of those).

Recent trends towards more expressive description logics like $\mathcal{SHIQ}$ [Horrocks et al., 2000] forced to revisit usage of theorem provers for description logics. Two directions can be observed in recent works. One of them [Hustadt, Motik & Sattler, 2004] describes a particular strategy based on a *basic superposition calculus* using which subsumption of concepts in $\mathcal{SHIQ}$ can be *decided*. However, it is not straightforward to implement such a strategy using existing theorem provers, since they provide only for a limited control over the saturation process. In particular it is not possible to specify custom orderings, selection functions or inference rules.

Another approach exemplified with a work of Tsarkov, Riazanov, Bechhofer & Horrocks [2004], suggests to use theorem prover "as is". Instead of trying to design special saturation strategies for a theorem prover, they suggest to put an effort towards finding good preprocessing techniques (e.g., removing of irrelevant axioms), which allow one to solve *most* of the subsumption tests *in practice*. Using this approach, a best state-of-the-art theorem prover (Vampire [Riazanov & Voronkov, 2002] in their tests) can be used to reason in whatever-expressive description logic which can be translated to first-order logic. Of course, this approach does not give a decision procedure for the concept subsumption problem, however, a prover can be used as an *incomplete* classifier for languages in which subsumption is undecidable (like Kl-One family), or for which no complete subsumption algorithm is known so far (like the ontology web language OWL).

The key difference between the above works and the approach presented in this chapter is that we, by no means, suggest to *use* a general theorem prover for reasoning in description logic. Instead of this, we advocate to use the *theory of resolution* and its extensions to *design* such reasoning algorithms.

As we have seen, some resolution strategies can be expressed in a more concise and efficient way, as datalog programs. This correspondence bridges description logic and logical programs, which allows one to apply all spectrum of formal tools available for logical programs – like query evaluation, optimisation and complexity analysis – to description logics. For example, one could try to prove formally that a datalog program specified by rules in Table 2.7 and Table 2.10 computes the same subsumption relation as the one given in (2.4)[9]. This would provide a correctness proof for the last program which is more efficient.

There have been many works on integration of description logics and database reasoning [see Borgida, Lenzerini & Rosati, 2003], however, most of them were concerned with *extensions* of description logics with some features from logical programs. Grosof, Horrocks, Volz & Decker [2003] suggest to identify common parts of logical programs and description logics which can be used as interfaces between these formalisms. However the fragment of description logic that can be embedded into datalog, called *description Horn logic* (short DHL), differs from $\mathcal{EL}$. In particular, inclusion axioms of form $A \sqsubseteq \exists R.B$ are not allowed, but axioms of form $A \sqsubseteq \forall R.B$ (but not $\forall R.B \sqsubseteq A$) are admissible. Basically, this fragment admits all inclusion axioms that can be translated to functional-free Horn clauses. However, DHL can be hardly used for expressing concept *definitions* because of this asymmetry in the usage of constructors in inclusion axioms.

Note that, although extensions of $\mathcal{EL}$ with many constructors become not tractable, some inclusion axioms involving them can be expressed in $\mathcal{EL}$, for example the following ones:

$$\begin{array}{lll}
C \sqcup D \sqsubseteq E & \dashrightarrow & C \sqsubseteq E,\ D \sqsubseteq E; \\
C \sqsubseteq \forall R^{-}.D & \dashrightarrow & \exists R.C \sqsubseteq D; \\
C \sqsubseteq \forall R.\bot & \dashrightarrow & C \sqcap \exists R.\top \sqsubseteq \bot; \\
C \sqcap \forall R.D \sqsubseteq \bot & \dashrightarrow & C \sqsubseteq \exists R.D_1,\ D_1 \sqcap D \sqsubseteq \bot; \\
C \sqsubseteq \exists R^{\geq n}.D & \dashrightarrow & C \sqsubseteq \exists R.(D \sqcap B_i),\ B_i \sqcap B_j \sqsubseteq \bot, \quad 1 \leq i < j \leq n;^{10} \\
\quad\ldots\ldots\ldots \text{ etc.} & &
\end{array} \tag{2.19}$$

It is well-known that *simplification rules* can dramatically improve the behaviour of saturation-based theorem provers. *Simplification rules* like Elimination of Duplicate Literals or Splitting through New Predicate Symbol, delete clauses that are no longer needed in a saturation process. For example, a resolution decision procedure for $\mathcal{EL}$ can be enchanted with Subsumption Deletion as follows:

$$\begin{array}{l}
\texttt{SD}[\mathsf{D4},\mathsf{D5}]\text{: } \neg A(x) \vee C(x),\ [\![\, \neg A(x) \vee \neg B(x) \vee C(x) \,]\!] \ \vdash \\
\texttt{SD}[\mathsf{D4},\mathsf{D5}]\text{: } \neg A(x) \vee C(x),\ [\![\, \neg B(x) \vee \neg A(x) \vee C(x) \,]\!] \ \vdash
\end{array}$$

These rules express that a clause is deleted if there is a proper subclause derived.

[9] Although the last problem, which is known as *query equivalence*, is undecidable for datalog programs in general [see Dantsin et al., 2001]

[10] Here correctness of the translation relies on the *tree-model property* for the description logic. It is also possible to give a similar translation which uses only $\lceil \log n \rceil$ new concept names

Such inferences correspond to logical programs with *deletion*:

$$\begin{aligned} &\leftarrow \mathsf{D4}(A, C),\ \llbracket\, \mathsf{D5}(A, B, C) \,\rrbracket. \\ &\leftarrow \mathsf{D4}(A, C),\ \llbracket\, \mathsf{D5}(B, A, C) \,\rrbracket. \end{aligned}$$

(which might be implemented using *default negation*). Logical programs with *priorities* and *deletion* were studied in [Ganzinger & McAllester, 2001, 2002], where certain complexity characterisations similar those of McAllester [2002] were derived.

As the next step, it might be reasonable to focus on PSPACE extensions of $\mathcal{EL}$ (with inverse roles, functionality and, partially, universal value restrictions), whose axioms correspond to Horn clauses, possibly with functional symbols. In fact, first-order translation of many ontologies, such as GALEN and TAMBIS correspond to such clauses [see Tsarkov et al., 2004].

Chapter 3

Preliminaries

This chapter introduces the standard material that will be used throughout this thesis. In section 3.1 we give an account on *first-order logic* and *term rewriting*, which underly the theory of saturation-based theorem proving. Then we give a brief introduction to modal languages (section 3.2) and first-order fragments (section 3.3) which will be the main sources of problems that we are going to address. In section 3.4 we introduce *Domino problems* which are the most common tool for proving undecidability results. Finally, in section 3.5, we introduce the framework of saturation-based theorem proving, where we formulate the most commonly used calculi in automated deduction and their variants. We make a particular emphasis on the usage of these calculi for decision procedures: we give a computational model of a theorem prover and estimate complexity of clause normal form transformations.

3.1 Logical Preliminaries

We assume the reader to have a background knowledge on first-order logic and term rewriting. A comprehensive introduction to these areas could be found in textbooks [Fitting, 1996] and [Baader & Nipkow, 1998]. Below we give some necessary notations and facts. In [Kazakov, 2005] the reader may find an extended version of the material given in this section.

3.1.1 First-Order Logic

Syntax of first-order logic

A *first-order signature* is a triple $\Sigma = (\mathrm{Pre}, \mathrm{Fun}, \mathrm{Var})$ consisting of a set of *predicate symbols* Pre, a set of *functional symbols* Fun and a set of *variables* Var. For every $p \in \mathrm{Pre}$ and $f \in \mathrm{Fun}$, there is a unique integer $ar(p)$, respectively $ar(f)$ called *the* *arity* of the symbol. The sets of (first-order) *terms* Tm_Σ and (first-oder) *atoms* At_Σ over Σ are defined inductively as follows:

$$\mathrm{Tm}_\Sigma ::= x \mid f(t_1,..,t_n) \ . \qquad \mathrm{At}_\Sigma ::= p(t_1,..,t_m) \ . \tag{3.1}$$

where $x \in \mathrm{Var}$, $f \in \mathrm{Fun}$, $p \in \mathrm{Pre}$, $n = ar(f)$, $m = ar(g)$, and $t_i, t_j \in \mathrm{Tm}_\Sigma$ for $1 \leq i \leq n$, $1 \leq j \leq m$. The set of *first-order formulas* Fm_Σ over Σ is defined using Boolean connectives and quantifiers as usual:

$$\mathrm{Fm}_\Sigma ::= A \mid F_1 \vee F_2 \mid F_1 \wedge F_2 \mid \neg F_1 \mid \forall y.F_1 \mid \exists y.F_1 \ . \tag{3.2}$$

where $A \in \mathrm{At}_\Sigma$, $F_i \in \mathrm{Fm}_\Sigma$, $i = 1, 2$, and $y \in \mathrm{Var}$. For conciseness we sometimes write $\between$ to denote either conjunction *or* disjunction, and Q to denote a quantifier. We also use $\overline{x}$ to represent a vector of variables $x_1,..,x_k$ for $k \geq 0$. For example, $Q\overline{x}.F$ denotes a first-order formula with some prefix of quantifiers.

A signature Σ may contain a distinguished binary predicate $\simeq$ which is called the *equality predicate*. In this case we deal with the *first-order logic with equality*. We use the *infix notation* for *equational atoms*: $s \simeq t$, which is not distinguished from $t \simeq s$. The *negation* of an equational atom is denoted by $s \not\simeq t$.

The *size* $|t|$, $|A|$, $|F|$, of a *term* t, *atom* A and *formula* F is its length. We use symbol $\trianglelefteq$ to denote the *subterm* and *subformula* relations on terms and formulas: by writing $s \trianglelefteq t$ and $s \trianglelefteq F$ we mean that s is a *subterm* of t, respectively of F, and $G \trianglelefteq F$ means that G is a *subformula* of F. Similarly $\triangleleft$ denotes the *strict* subterm and subformula relations, and $\trianglerighteq$ and $\triangleright$ are their inverses. Given a first-order formula F we denote by $\mathit{free}[F]$ the set of *free variables* of F and by $\mathit{vars}[F]$ the set of *all variables* from F ($\mathit{free}[F] \subseteq \mathit{vars}[F]$). The *width* $\mathit{width}(F)$ of a formula F is the maximal number of free variables in a subformula of F.

We write $F[G]$ ($F[s]$, $h[s]$) to denote a respective formula or a term with *indicated occurrences* of its subformula G (or its subterm s). $F[G/H]$ ($F[s/t]$, $h[s/t]$) denotes the result of replacing all these occurrences by a formula H (term t). When replaced occurrences are clear from the context, we shorten this to $F[H]$ ($F[t]$, $h[t]$).

Semantics of first-order logic

Given a signature Σ, a *first-order interpretation* (sometimes called a Σ-*structure*) is a pair $\mathcal{I} = (\boldsymbol{D}, \cdot^{\mathcal{I}})$, where $\boldsymbol{D}$ is a non-empty set called the *domain* of the interpretation, and $\cdot^{\mathcal{I}}$ is a mapping that associates (***i***) to every functional symbol $f \in \mathrm{Fun}$ with $n = ar(f)$ a function $f^{\mathcal{I}} : \boldsymbol{D}^n \to \boldsymbol{D}$; (***ii***) to every non-equality predicate symbol $p \in \mathrm{Pre} \setminus \{\simeq\}$ with $m = ar(p)$ a relation $p^{\mathcal{I}} \subseteq \boldsymbol{D}^m$. A *(variable) valuation* is a mapping $\eta : \mathrm{Var} \to \boldsymbol{D}$. For any $x \in \mathrm{Var}$ and $d \in \boldsymbol{D}$, let $\{x \mapsto d\}{\cdot}\eta$ denote the valuation for which $\eta'(x) = d$ and $\eta'(y) = \eta(y)$ for $x \neq y$. The *value* $[t]^{\mathcal{I}}_{\eta} \in \boldsymbol{D}$ of a *term* $t \in \mathrm{Tm}_\Sigma$ and the *truth value* $[F]^{\mathcal{I}}_{\eta} \in \{\texttt{true}, \texttt{false}\}$ of a formula $F \in \mathrm{Fm}_\Sigma$ under an interpretation $\mathcal{I}$ with a valuation η are defined as usual.

A first-order formula F is *satisfiable in an interpretation* $\mathcal{I}$, if there exists a valuation η such that $[F]^{\mathcal{I}}_{\eta} = \texttt{true}$. In this case $\mathcal{I}$ is a *model* for F. A formula F is *satisfiable* if there is a model $\mathcal{M}$ for F. A formula F is *valid in an interpretation* $\mathcal{I}$ (notation: $\mathcal{I} \vDash F$, if $[F]^{\mathcal{I}}_{\eta} = \texttt{true}$ for *every* valuation η. A formula F is *valid* ($\vDash F$) if F is valid in every interpretation $\mathcal{I}$. A formula G is a *logical consequence* of a formula F (notation: $F \vDash G$), if for every interpretation $\mathcal{I}$ and valuation η, $[F]^{\mathcal{I}}_{\eta} = \texttt{true}$ implies that $[G]^{\mathcal{I}}_{\eta} = \texttt{true}$. A formula G is *(logically) equivalent* to F (notation: $G \equiv F$) if both formulas are logical consequences of each other. Formulas F and G are *equisatisfiable* when F is satisfiable *iff* G is satisfiable.

A *theory* $\boldsymbol{T}$ is a collection of first-order formulas that are called the *theory axioms*. A $\boldsymbol{T}$*-interpretation* is an interpretation that satisfies all axioms of $\boldsymbol{T}$. We say that a formula F is $\boldsymbol{T}$*-satisfiable* ($\boldsymbol{T}$*-valid*) if F is satisfiable in some (respectively all) $\boldsymbol{T}$-interpretations.

We will often extend signatures by adding new predicate or functional symbols. This requires modification of interpretations in such a way that satisfiability of formulas over the old signature is preserved.

Definition 3.1.1 (Conservative)**.** A signature $\Sigma' = (\mathrm{Pre}', \mathrm{Fun}', \mathrm{Var}')$ is called an *extension* of a signature $\Sigma = (\mathrm{Pre}, \mathrm{Fun}, \mathrm{Var})$, if $\mathrm{Pre} \subseteq \mathrm{Pre}'$, $\mathrm{Fun} \subseteq \mathrm{Fun}'$ and $\mathrm{Var} \subseteq \mathrm{Var}'$. In such situation, we say that a Σ'-interpretation $\mathcal{I}' = (\boldsymbol{D}', \cdot^{\mathcal{I}'})$ is an *expansion* of a Σ-interpretation $\mathcal{I} = (\boldsymbol{D}, \cdot^{\mathcal{I}})$, if $(\boldsymbol{i})$ $\boldsymbol{D} = \boldsymbol{D}'$ and $(\boldsymbol{ii})$ $f^{\mathcal{I}'} = f^{\mathcal{I}}$, $p^{\mathcal{I}'} = p^{\mathcal{I}}$ for every functional symbol $f \in \mathrm{Fun} \subseteq \mathrm{Fun}'$ and every predicate symbol $p \in \mathrm{Pre} \subseteq \mathrm{Pre}'$.

We say that a formula F' is *conservative over* a formula F if $(\boldsymbol{i})$ F is a logical consequence of F' and $(\boldsymbol{ii})$ every model of F can be expanded to a model of F'. ◈

3.1.2 First-Order Clause Logic

Most automated theorem provers (ATPs) for first-order logic do not operate directly with formulas, but with their simpler clause normal forms. A *(first-order) literal* L is an atom A or a negation of an atom $\neg A$. Two literals A and $\neg A$ are said to be *complementary.* Lt_Σ denotes the set of all literals constructed over a signature Σ. A *clause* is a disjunction of literals $C = L_1 \vee \cdots \vee L_k$. The set of all clauses is denoted by Cl_Σ. A clause C is interpreted as the first-order formula $\forall \overline{x}.C$, where $\overline{x}$ are all variables of C: $\overline{x} = \mathit{vars}[C]$. In other words, all variables of a clause C are *implicitly universally quantified.* Hence, C is $\texttt{true}$ in an interpretation $\mathcal{I}$, if $\mathcal{I} \vDash \forall \overline{x}.C$. A *clause set* $N \subseteq \mathrm{Cl}_\Sigma$ is $\texttt{true}$ in an interpretation $\mathcal{I}$ if every clause C from N is $\texttt{true}$ in $\mathcal{I}$.

A *term/atom/literal or* a *clause* is called *ground* if it contains no variables. We assume that a first-order signature $\Sigma = (\mathrm{Pre}, \mathrm{Fun}, \mathrm{Var})$ contains at least one constant (otherwise we add some fixed constant c_0), so the set Tm^0_Σ of *ground terms* over Σ is not empty: $\{\} \neq \mathrm{Tm}^0_\Sigma \subseteq \mathrm{Tm}_\Sigma$. The sets of ground atoms and ground literals are denoted respectively by At^0_Σ and Lt^0_Σ.

By an *expression* E we mean a term or a literal. An *expression symbol* e is either a functional symbol f or a predicate symbol p or a *negated* predicate symbol $\neg p$. In the last two cases we deal with a *literal symbol* l. Sometimes we will form expressions by attaching a *sequence of arguments* $(t_1,..,t_n)$ to an expression symbol e: $E = e(t_1,..,t_n)$, where $t_i \in \mathrm{Tm}_\Sigma$, $1 \leq i \leq n = ar(e)$. In this case we say also that $(t_1,..,t_n)$ are the *arguments of* E.

The *size* $|E|$, $|C|$ *of an expression* E *or a clause* C is determined by treating them as appropriate terms or formulas. The *depth* $depth(E)$ of an expression E is defined as follows: $depth(x) := 1$; $depth(e(t_1,..,t_n)) := 1 + \max\{0, depth(t_1),.., depth(t_n)\}$.

An expression E is *shallow*, if $depth(E) \leq 2$, i.e., all arguments of the expression are variables or constants. A literal L is *simple* if $depth(L) \leq 3$, i.e., all its arguments are shallow. An expression E or a clause C is *functional* if it contains at least one functional symbol.

3.1.3 Orderings

A *(strict partial) ordering (or order)* $\succ$ on a set $\boldsymbol{D}$ is a transitive and irreflexive binary relation on $\boldsymbol{D}$. If $\succ$ is a strict ordering then its reflexive closure is denoted by $\succeq$. An ordering $\succ$ is *total* or *linear* if every two different elements are *comparable* by $\succ$, i.e., for every $d_1, d_2 \in \boldsymbol{D}$, $d_1 \neq d_2$ implies that either $d_1 \succ d_2$ or $d_2 \succ d_1$. An ordering $\succ$ is *well-founded* or *Noetherian* if there is no infinite descending chain $d_1 \succ d_2 \succ \cdots$ of elements $d_i \in \boldsymbol{D}$, $i \geq 1$. A total well-founded order is called a *well-order*.

A *quasi-ordering* $\succsim$ on $\boldsymbol{D}$ is any reflexive and transitive relation on $\boldsymbol{D}$. An *equivalence relation induced* by a quasi-ordering $\succsim$ is the *symmetrical part* $\sim$ of $\succsim$: $d_1 \sim d_2$ *iff* $d_1 \succsim d_2$ and $d_2 \succsim d_2$. A *strict part* $\succ$ of a quasi-ordering $\succsim$ is the difference between $\succsim$ and $\sim$: $d_1 \succ d_2$ *iff* $d_1 \succsim d_2$ and $d_2 \not\succsim d_1$. Note that the strict part $\succ$ of $\succsim$ is the greatest ordering contained in $\succsim$.

Multiset and lexicographic extensions of orderings

A *multiset* of elements from $\boldsymbol{D}$ is a function $M : \boldsymbol{D} \to \mathbb{N}$. The number $M(d)$ is called the *multiplicity* of an element d in M, $d \in \boldsymbol{D}$. The *size* $|M|$ of a multiset M is defined by $|M| := \sum_{d \in D} M(d)$. A multiset M is *finite* if $|M| < \infty$.

Any ordering $\succ$ on $\boldsymbol{D}$ can be extended to an ordering $\succ_{mul}$ on finite multisets of $\boldsymbol{D}$ as follows: $M_1 \succ_{mul} M_2$ *iff* $(\boldsymbol{i})$ $M_1 \neq M_2$ and $(\boldsymbol{ii})$ for every element $d \in \boldsymbol{D}$, either $M_1(d) \geq M_2(d)$, or, otherwise there exists for some $d' \succ d$, $d' \in \boldsymbol{D}$, such that $M_1(d') > M_2(d')$. The ordering $\succ_{mul}$ is called the *multiset extension of* of the ordering $\succ$.

Any ordering $\succ$ on $\boldsymbol{D}$ can be extended to an ordering $\succ^n_{\text{lex}}$ on $\boldsymbol{D}^n$ called the *lexicographic extension of* of $\succ$ as follows: $(d_1,..,d_n) \succ^n_{\text{lex}} (d'_1,..,d'_n)$ *iff* there exists i with $1 \leq i \leq n$ such that $d_i \succ d'_i$, and for all j with $1 \leq j < i$, we have $d_j = d'_j$.

Reduction orderings

Let $\succ$ be an ordering on ground expressions over a signature Σ (i.e., on ground terms and ground literals). We say that $\succ$ is a *rewrite ordering* if $\succ$ admits the monotonicity property: for every ground terms s^0, t^0 and a ground expression E^0 with $s^0 \lhd E^0$, $s^0 \succ t^0$ implies $E^0[s^0] \succ E^0[t^0/s^0]$. A *reduction ordering* is a well-founded rewrite ordering. An ordering $\succ$ has the *subterm property* if for every $t^0 \lhd E^0$ we have $E^0 \succ t^0$. A *simplification ordering* is any reduction ordering with the subterm property.

Most term orderings used in applications nowadays are variations of either Knuth-Bendix ordering [Knuth & Bendix, 1970] or a lexicographic path ordering [Kamin & Lévy, 1980]. Both orderings are based on a *precedence* $\gg$, which is a strict order on functional symbols Fun of a signature Σ.

The Knuth-Bendix ordering A *weight function* is any function *weight* : Fun $\rightarrow \mathbb{N}$ that assigns a non-negative integer[1] to every functional symbol from Fun. A weight function *weight*$(\cdot)$ is *admissible* for a precedence $\gg$ *iff* (***i***) *weight*$(c) > 0$ for every constant c and (***ii***) for every *unary* functional symbol $f \in$ Fun, *weight*$(f) = 0$ implies that f is $\gg$-greatest element in Fun (i.e., for every $g \in \text{Fun} \setminus \{f\}$, we have $f \gg g$). The weight function is recursively extended to the set of ground terms Tm^0_Σ as follows: $\mathit{weight}(f(t^0_1,..,t^0_n)) := \mathit{weight}(f) + \mathit{weight}(t^0_1) + \cdots + \mathit{weight}(t^0_n)$. Note, that if *weight*$(f) = 1$ for every functional symbol $f \in$ Fun, then *weight*$(t) = |t|$, where $|t|$ is the size of t.

Definition 3.1.2. The *Knuth-Bendix ordering* (short *KBO*), induced by a precedence $\gg$ and an admissible weight function *weight*$(\cdot)$ is defined as follows: For every pair of ground terms $s^0 = f(s^0_1,..,s^0_n)$ and $t^0 = g(t^0_1,..,t^0_m)$ we have $s^0 \succ_{kbo} t^0$ *iff* one of the following conditions holds:

(**1**) $\mathit{weight}(s^0) > \mathit{weight}(t^0)$, or
(**2**) $\mathit{weight}(s^0) = \mathit{weight}(t^0)$, but $f \gg g$, or

[1] some definitions, e.g., in [Baader & Nipkow, 1998], allow for non-negative real weights, however the advantage of this is doubtful

(**3**) $weight(s^0) = weight(t^0)$, $f = g$ (and hence $m = n$), and $(s^0_1,..,s^0_n) \succ^{kbo}_{\text{lex}} (t^0_1,..,t^0_n)$, *where* $\succ^{kbo}_{\text{lex}}$ *is the lexicographic extension of* $\succ_{kbo}$. ◈

Theorem 3.1.3 ([see Baader & Nipkow, 1998, Theorem 5.4.20])**.** *Let* $\gg$ *be a precedence on functional symbols* Fun *of a signature* Σ *and* $\mathit{weight}(\cdot)$ *be an admissible weight function for* $\gg$*. Then the Knuth-Bendix order* $\succ_{kbo}$ *induced by* $\gg$ *and* $\mathit{weight}(\cdot)$ *is a simplification ordering.*

Proposition 3.1.4 ([see Kazakov, 2005, Proposition 2.18])**.** *Let* $\succ_{kbo}$ *be the Knuth-Bendix order induced by a total precedence* $\gg$ *and a weight function* $\mathit{weight}(\cdot)$*. Then* $\succ_{kbo}$ *is a total ordering.*

The lexicographic path ordering A reduction ordering can be defined based on a precedence of functional symbols only:

Definition 3.1.5. The *lexicographic path ordering* (short *LPO*), induced by a precedence $\gg$ is defined as follows: For every pair of ground terms $s^0 = f(s^0_1,..,s^0_n)$ and $t^0 = g(t^0_1,..,t^0_m)$ we have $s^0 \succ_{lpo} t^0$ *iff* one of the following conditions holds:

(**1**) $s^0_i \succeq_{lpo} t^0$ for some i with $1 \leq i \leq n$, or

(**2**) $f \gg g$ and $s^0 \succ_{lpo} t^0_j$ for all j with $1 \leq j \leq m$, or

(**3**) $f = g$ (and hence $m = n$), and $(s^0_1,..,s^0_n) \succ^{lpo}_{\text{lex}} (t^0_1,..,t^0_n)$, *where* $\succ^{lpo}_{\text{lex}}$ *is the lexicographic extension of* $\succ_{lpo}$. ◈

Analogs of Theorem 3.1.3 and Proposition 3.1.4 can be shown for *LPO*-orderings:

Theorem 3.1.6 ([see Baader & Nipkow, 1998, Theorem 5.4.14])**.** *For any precedence* $\gg$ *on functional symbols* Fun*, the ordering* $\succ_{lpo}$ *induced by* $\gg$ *is a simplification ordering on* Tm^0_Σ.

Proposition 3.1.7 ([see Kazakov, 2005, Proposition 2.21])**.** *Let* $\gg$ *be a total precedence on* Fun*, then the ordering* $\succ_{lpo}$ *induced by* $\gg$ *is a total ordering on* Tm^0_Σ.

KBO and *LPO* orders can be used for ground expressions by treating predicate symbols and negation as functional symbols (i.e., by defining precedence and weight functions on them). In the following example, we demonstrate a difference between *KBO* and *LPO*-orderings, that is important for saturation-based decision procedures.

Example 3.1.8. Consider two atoms: $\mathtt{p}(t^0, t^0)$ and $\mathtt{q}(\mathtt{f}(t^0))$, where $\mathtt{p}$ and $\mathtt{q}$ are predicate symbols, $\mathtt{f}$ is a functional symbol and t^0 is some ground term. Let $\gg$ be a precedence on predicate and functional symbols such that $\mathtt{f} \gg \mathtt{p}$ and let $\succ_{lpo}$ be the

LPO-ordering induced by $\gg$. Then we have $\mathtt{q}(\mathtt{f}(t^0)) \succ_{lpo} \mathtt{p}(t^0, t^0)$. Indeed, $t^0 \succeq_{lpo} t^0$, therefore by condition (**1**) from Definition 3.1.5 we have $\mathtt{f}(t^0) \succ_{lpo} t^0$. Since $\mathtt{f} \gg \mathtt{p}$, by condition (**2**) we have $\mathtt{f}(t^0) \succ_{lpo} \mathtt{p}(t^0, t^0)$ which yields again by condition (**1**) that $\mathtt{q}(\mathtt{f}(t^0)) \succ_{lpo} \mathtt{p}(t^0, t^0)$. Note, that this holds for *every* ground term t^0.

However it is not possible to have $\mathtt{q}(\mathtt{f}(t^0)) \succ_{kbo} \mathtt{p}(t^0, t^0)$ for all terms t^0 using a *KBO*-ordering $\succ_{kbo}$. Indeed, for every admissible weight function *weight*$(\cdot)$, one can construct a term t^0 with a large enough weight so that $\mathit{weight}(\mathtt{p}(t^0, t^0)) = \mathit{weight}(\mathtt{p}) + 2{\cdot}\mathit{weight}(t^0) > \mathit{weight}(\mathtt{q}) + \mathit{weight}(\mathtt{f}) + \mathit{weight}(t^0) = \mathit{weight}(\mathtt{q}(\mathtt{f}(t^0)))$. ⟐

3.1.4 Substitutions And Unification

A *substitution* is a function that maps variables to terms $\sigma : \mathrm{Var} \to \mathrm{Tm}_\Sigma$, which is denoted by $\sigma = \{x_1/t_1, x_2/t_2, .., x_n/t_n, ..\}$, or, shortly $\sigma = \{\overline{x}/\overline{t}\}$ (hereby $\sigma(x_i) = t_i$). The *domain* of a substitution $\mathit{Dom}(\sigma) := \{x_i \in \mathrm{Var} \mid \sigma(x_i) \neq x_i\}$. The *range* of a substitution $\mathit{Ran}(\sigma) := \{\sigma(x_i) \mid x_i \in \mathit{Dom}(\sigma)\}$. A substitution σ is a *renaming*, if (***i***) $\mathit{Ran}(\sigma) \subseteq \mathrm{Var}$ and (***ii***) $x \neq y$ implies $\sigma(x) \neq \sigma(y)$. A substitution σ^0 is *ground iff* $\sigma(x) \in \mathrm{Tm}^0_\Sigma$ for every $x \in \mathrm{Var}$.[2]

Given an expression E and a substitution σ, we denote by $E{\cdot}\sigma$ the *application* of σ to E which is computed by simultaneously applying all replacement of the substitution. A *composition of substitutions* σ_1 and σ_2 is a new substitution denoted by $\sigma_1{\cdot}\sigma_2$ that is defined by $(\sigma_1{\cdot}\sigma_2)(x) := (\sigma_1(x)){\cdot}\sigma_2$.

A *unification problem*[3] is a set $P = \{E_1{=}E'_1, .., E_n{=}E'_n\}$ of equations between expressions, $n \geq 0$. A *solution* for a unification problem P, called a *unifier* is a substitution σ such that $E_i{\cdot}\sigma = E'_i{\cdot}\sigma$ for every i with $1 \leq i \leq n$. A *most general unifier* (or, shortly *mgu*) for the unification problem P is a unifier σ such that for any other unifier σ' we have $\sigma' = \sigma{\cdot}\tau$ for some substitution τ. Note that this definition implies that a most general unifier is unique up to a renaming. There is an effective procedure that given a unification problem P, computes its most general unifier $\mathit{mgu}(P)$. See e.g., [Baader & Nipkow, 1998] or [Kazakov, 2005] for further details. When P consists of one equation $P = \{E{=}E'\}$, we usually write $\mathit{mgu}(E, E')$ instead of $\mathit{mgu}(\{E{=}E'\})$.

Covering Expressions and Atomic Substitutions

The notion of covering expressions has been introduced by Fermüller et al. [1993] to describe resolution decision procedures for certain clause classes. The following

[2] It is generally assumed that every substitution has a finite domain, which we do not require in this thesis, since otherwise it is more tricky to define the notion of a ground substitution

[3] In this thesis we are concerned only with the *syntactic* unification problems

notions and results originate from [Fermüller et al., 1993]. Full proofs for the given propositions can be found in [Kazakov, 2005].

Definition 3.1.9 (Atomic)**.** A term is *atomic* if it is either a constant or a variable. A substitution σ is *atomic on a set of variables* V, if for every $x \in V$, $\sigma(x)$ is atomic. A substitution σ is *atomic* if σ is atomic on Var (equivalently on $\mathit{Dom}(\sigma)$). ◇

The nice property of atomic substitutions is that they do not change the depth of the expression when applied to it:

Proposition 3.1.10. *Let E be an expression and σ be a substitution that is atomic on* $\mathit{vars}[E]$*. Then* $\mathit{depth}(E{\cdot}\sigma) = \mathit{depth}(E)$*.*

Definition 3.1.11 (Covering)**.** An expression E *covers a set of variables V* (notation: $E \propto V$) *iff* every non-atomic subterm t of E contains all variables form V ($V \subseteq \mathit{vars}[t]$). An expression E_1 *covers an expression E_2* (or a clause C) (notation: $E_1 \propto E_2$, $E_1 \propto C$) *iff* E_1 covers $\mathit{vars}[E_2]$ ($\mathit{vars}[C]$). An expression E is *covering iff* $E \propto E$. A clause C is covering *iff* all literals from C cover C. ◇

Example 3.1.12. The following expressions cover the set of variables $V = \{\mathtt{x}, \mathtt{y}\}$:
$\mathtt{a(c, f(x, y))}$; $\mathtt{a(h(x, x, c, y), y)}$; $\mathtt{a(h(x, x, f(x, y), f(y, x)), c)}$; $\mathtt{a(x, c)}$; $\mathtt{x}$; $\mathtt{c}$;
$\mathtt{z}$; $\mathtt{a(x, z)}$; $\mathtt{a(c, h(x, z, c, y), y)}$ and $\mathtt{a(f(x, y), z)}$.
All expressions in the first line cover each other. Every expression in the second line covers every expression in the first line. The last expression covers none of the expressions in the last line. The expression before, covers all expressions. All but the last expressions are covering.

The following expressions do not cover $V = \{x, y\}$: $\mathtt{a(g(x), x)}$; $\mathtt{a(c, f(y, c))}$; $\mathtt{a(x, f(x, z))}$; $\mathtt{a(f(x, y), g(z))}$; $\mathtt{a(g(c), f(x, y))}$; and $\mathtt{a(f(x, x), y)}$.
However first three of them are covering. First two and last two expressions are covered by every expression from the first group. ◇

Covering expression typically appear in the result of Skolemization for relational first-order formulas. The class of covering expressions can be extended to so-called *weakly covering* expressions. This notion has been used for defining many decidable clause classes, including $\mathcal{E}^+$ [Fermüller et al., 1993] and a clause class for the guarded fragment [de Nivelle, 1998; de Nivelle & de Rijke, 2003]. We will not make use of weakly covering expressions in this thesis, but we include their definition for the sake of completeness.

Definition 3.1.13 (Weakly Covering)**.** An expression E *weakly covers* a set of variables V, *iff* every non-ground subterm t of E contains all variables from V ($V \subseteq \mathit{vars}[t]$). An expression E_1 *weakly covers* an expression E_2 (or a clause C), *iff* E_1 weakly covers $\mathsf{vars}[E_2]$ ($\mathsf{vars}[C]$). An expression E is *weakly covering iff* E weakly covers E. A clause C is *weakly covering iff* all literals in C weakly cover C. ◇◇

The following lemma plays an important role for proving decidability of some clause classes by resolution:

Lemma 3.1.14. *Let* E_1 *and* E_2 *be covering expressions such that* $depth(E_1) \geq depth(E_2)$*. Let* $\sigma = mgu(E_1{=}E_2)$*. Then* σ *is atomic on* $\mathsf{vars}[E_1]$*.*

The lemma essentially says that unification of covering expressions does not enlarge their maximal depth, since by Proposition 3.1.10, $depth(E_2{\cdot}\sigma) = depth(E_1{\cdot}\sigma) = depth(E_1)$. This argument can be used in saturation-based decision procedures for showing that the depth of generated clauses does not grow beyond a certain limit.

3.2 Modal and Description Logics

3.2.1 Propositional Modal Logics

In this section we give a brief introduction to propositional modal logic. There are numerous introductory books on this topic [e.g., Goldblatt, 1987; Hughes & Cresswell, 1996; Chagrov & Zakharyaschev, 1997; Blackburn, de Rijke & Venema, 2001].

Modal logics have their roots in works of philosophers, starting from Aristotle, who were trying to axiomatise different kinds of truth. Formal treatment of *non-classical logics* began in the beginning of 20th century from works of Lewis, who proposed several axiomatic system for a notion of *strict implication*. Modal logics in their modern syntax are known after Gödel [1933*a*].

Modal logic extends propositional logic with additional operator $\Box(\cdot)$, where formula $\Box F$ is read, depending on application, as "*F is necessarily true*", or "*F is always true*" or "*F is known*" or "*F is provable*". Formally, the set of *modal formulas* can be defined by an abstract grammar:

$$\mathrm{MF} \ ::= \ A \mid \bot \mid \neg F_1 \mid F_1 \wedge F_2 \mid \Box F_1 \ . \tag{3.3}$$

where A is a *propositional atom*, and $F_1, F_2 \in \mathrm{MF}$ are *modal formulas*. Other *Boolean operators*, e.g., $\rightarrow$, $\vee$,and $\leftrightarrow$ can be expressed in the usual way. In addition, one usually introduces an operator $\Diamond(\cdot)$ that is dual to $\Box(\cdot)$, i.e., $\Diamond F$ is an abbreviation for $\neg\Box\neg F$.

The *basic modal logic* $\mathbf{K}$ is defined by augmenting a set of axioms for propositional logic with an additional *axiom schemata*:

$$\text{K.}\quad \Box(X \to Y) \to (\Box X \to \Box Y)$$

where X and Y are arbitrary modal formulas. This axiom is often called *normality axiom* and modal logics that contain it are called *normal modal logics*. The resulted axiom system is closed under two inference rules:

Modus Ponens (MP): $X;\ X \to Y \vdash Y$

Necessitation (NEC): $X \vdash \Box X$

A modal formula F that can be derived in this system (in symbols $\mathbf{K} \vdash F$) is called a *theorem* of $\mathbf{K}$. Depending on reading of the modal operators, different axiom schemata have been proposed that extend the basic modal logic $\mathbf{K}$. The most common from them are listed below:

$$\begin{array}{llllll} \text{D.} & \Box X \to \Diamond X & \text{B.} & X \to \Box\Diamond X & 4. & \Box X \to \Box\Box X \\ \text{T.} & \Box X \to X & & & 5. & \Diamond X \to \Box\Diamond X \end{array} \tag{3.4}$$

Extensions of modal logics are usually called by listing their axiom names, for example, modal logic **KT4** is an extension of **K** with axioms T and 4, which has an alternative name **S4**.

The main reasoning problem for modal logics is to decide whether a given modal formula F is provable in the respective system. The most widely used *tableau procedures* for solving this problem originate from works of Kripke [1959], who introduced *models* for modal logics.

Definition 3.2.1. A *Kripke interpretation* for modal logic is a triple $\mathcal{I} = (W, R, v)$, where W is a non-empty set of *possible worlds*, $R \subseteq W \times W$ is a binary relation on W called an *accessibility relation*, and $v : W \to 2^{\mathrm{At}}$ is a function that assigns to every world $w \in W$, a set of propositional atoms. A *truth value* $[F]^{\mathcal{I}}_{w} \in \{\texttt{true}, \texttt{false}\}$ of a modal formula F in world w under an interpretation $\mathcal{I}$ is computed inductively over (3.3) as follows:

$$\begin{array}{rrcllll} [F]^{\mathcal{I}}_{w} := & [A]^{\mathcal{I}}_{w} & = & \texttt{true} & \textit{iff} & A \in v(w) & \mid \\ & [\bot]^{\mathcal{I}}_{w} & = & \texttt{false} & & & \mid \\ & [\neg F_1]^{\mathcal{I}}_{w} & = & \texttt{true} & \textit{iff} & [F_1]^{\mathcal{I}}_{w} = \texttt{false} & \mid \\ & [F_1 \wedge F_2]^{\mathcal{I}}_{w} & = & \texttt{true} & \textit{iff} & [F_1]^{\mathcal{I}}_{w} = \texttt{true} \quad \textit{and} \quad [F_2]^{\mathcal{I}}_{w} = \texttt{true} & \mid \\ & [\Box F_1]^{\mathcal{I}}_{w} & = & \texttt{true} & \textit{iff} & [F_1]^{\mathcal{I}}_{w'} = \texttt{true} \quad \textit{for all} \quad w' \textit{ with } (w, w') \in R\ . & \end{array}$$

A formula F is *satisfiable in an interpretation* $\mathcal{I} = (W, R, v)$, if there exists $w \in W$ such that $[F]^{\mathcal{I}}_{w} = \texttt{true}$. F is *valid* in $\mathcal{I}$ (notation $\mathcal{I} \vDash F$), if for every $w \in W$, we have $[F]^{\mathcal{I}}_{w} = \texttt{true}$. ◇◇

The semantics for modal logic given in Definition 3.2.1 is called *Kripke, or possible world semantics*. It is complete for modal logic **K** in the following sense:

Proposition 3.2.2 (Completeness of **K**). *A model formula F is a theorem of* **K** *if and only if F is valid in every Kripke interpretation $\mathcal{I}$ (in symbols: $\mathbf{K} \vdash F$ iff $I \vDash F$ for every $\mathcal{I}$).*

Kripke semantics for modal formulas can be naturally encoded in first order logic, using a so-called *relational translation*. To define this translation, we assign to every propositional atom A, a unary relation $A(x)$, which is understood as "*A is* `true` *in world x*". A reachability relation R corresponds to a binary relation $R(x, y)$. This assignment is extended using Definition 3.2.1, to arbitrary modal formulas: for every modal formula F, we inductively define a first-order formula $\tau(F, x)$ which expresses that "*F is* `true` *in wold x*":

$$\begin{aligned} \tau(F,x) \;:=\quad \tau(A,x) &= A(x) & \mid \\ \tau(\bot) &= \bot & \mid \\ \tau(\neg F_1,x) &= \neg\tau(F_1,x) & \mid \\ \tau(F_1 \wedge F_2,x) &= \tau(F_1,x) \wedge \tau(F_2,x) & \mid \\ \tau(\Box F_1,x) &= \forall y.[R(x,y) \rightarrow \tau(F_1,y)] & . \end{aligned} \tag{3.5}$$

For example, an instance $\Box(A \rightarrow B) \rightarrow (\Box A \rightarrow \Box B)$ of axiom scheme K is translated according to definition (3.5) to a valid first-order formula:

$$\forall y.[R(x,y) \rightarrow (A(y) \rightarrow B(y))] \;\rightarrow\; (\forall y.[R(x,y) \rightarrow A(y)] \;\rightarrow\; \forall y.[R(x,y) \rightarrow B(y)])$$

(assuming the convention for implication). In general, it can be observed that function (3.5) maps modal formulas into a *modal fragment* of $\mathcal{FO}$, which is defined as follows:

$$\mathcal{FO}(\mathrm{MF}) \;::=\; A(x) \mid \bot \mid \neg F_1 \mid F_1 \wedge F_2 \mid \forall y.[R(x,y) \rightarrow F_1[y]] \;. \tag{3.6}$$

where $A(x)$ is a unary atom, $R(x, y)$ is a binary atom and F_1, $F_2 \in \mathcal{FO}(\mathrm{MF})$.

To define models for extensions of basic modal logic **K**, one usually restricts reachability relations to those that satisfy certain properties called *frame correspondence properties*. In Table 3.1 we have listed the correspondence properties for modal axioms (3.4). Not every modal axiom schemata corresponds to a first-order definable property of the reachability relation, and obtaining such properties is a non-trivial task [see Ohlbach, 1996; Ohlbach & Schmidt, 1997].

Apart from the *relational translation*, a variety of other translational methods for modal formulas have been proposed in literature, notably the *(optimized) functional translation* [Ohlbach, 1996; Schmidt, 1997] (see also [Ohlbach, Nonnengart, de Rijke & Gabbay, 2001] for an overview of those methods).

Table 3.1 Frame correspondence properties for some modal axioms

	Axiom	*Correspondence Property*	First-order axiom
D.	$\Box X \to \Diamond X$	*Serial*	$\forall x.\exists y.R(x,y)$
T.	$\Box X \to X$	*Reflexive*	$\forall x.R(x,x)$
B.	$X \to \Box\Diamond X$	*Symmetric*	$\forall xy.[R(x,y) \to R(y,x)]$
4.	$\Box X \to \Box\Box X$	*Transitive*	$\forall xyz.[R(x,y) \wedge R(y,z) \to R(x,z)]$
5.	$\Diamond X \to \Box\Diamond X$	*Euclidean*	$\forall xyz.[R(x,y) \wedge R(x,z) \to R(y,z)]$

The language of modal logics can be extended to several modalities $\Box_i$ with $i \in I$ for some set of indices I. A *multi-modal* version $\mathbf{K_n}$ of the basic modal logic $\mathbf{K}$, is defined by taking the normality axiom and necessitation rule for every $\Box_i$. The possible world semantics for $\mathbf{K_n}$ is defined using several reachability relations that correspond to each modality. The first-order translation (3.5) can be straightforwardly extended to such multi-modal case.

del Cerro & Panttonen [1988], Baldoni et al. [1998] and Demri [2001] have considered classes of so-called *inclusion multi-modal logics* characterised by *interaction axioms* of form $\Box_{i_1} \dots \Box_{i_n} X \to \Box_{j_1} \dots \Box_{j_m} X$ (which are equivalent to $\Diamond_{j_1} \dots \Diamond_{j_m} X \to \Diamond_{i_1} \dots \Diamond_{i_n} X$). These axioms correspond to frame properties $R_{j_1} \circ \dots \circ R_{j_m} \subseteq R_{i_1} \circ \dots \circ R_{i_n}$, where $\circ$ is a usual composition of binary relations. Most of these modal logics appear to be undecidable, however Baldoni et al. [1998] and Demri [2001] identified some decidable classes of inclusion modal logics, whose interaction axioms correspond to production rules for *regular languages*. In this thesis we cover some of these results using relational translation and saturation-based theorem proving.

3.2.2 Description Logics

In chapter 2 we already gave a brief introduction to description logics and related notions on the example of a simple terminological language $\mathcal{EL}$. In this section we repeat some general definitions for description logics and define their semantics and first-order translation. A comprehensive introduction to description logics can be found in the description logic handbook [Baader, Calvanese, McGuinness, Nardi & Patel-Schneider, 2003].

Description logics (short DL) originate from *semantic networks* and *frames* [see Brachman, 1979], which were developed to support a *(schematic) representation* of information. Although such kinds of systems are still popular (e.g., UML-diagrams for software development), it has been soon realised that those systems are inappropriate for *formal reasoning* because of the *ambiguity* in interpretation of such information, or, in other words, the lack of *formal semantics*. Consequently, presence of well-defined formal semantics, became the main distinguished feature of description logics [Levesque & Brachman, 1987].

We introduce a *basic description logic* $\mathcal{ALC}$ [Schmidt-Schauß & Smolka, 1991], which is traditionally assumed to be the minimal language required for conceptual reasoning.[4] A language of $\mathcal{ALC}$ is defined from a set of *concept names* CN and a set of *role names* RN by the following grammar:

$$\begin{array}{rrl} \mathcal{ALC} ::= & A \mid & \text{- \textit{concept name (atomic concept)}} \\ & \bot \mid & \text{- \textit{bottom concept}} \\ & \neg C_1 \mid & \text{- \textit{negation}} \\ & C_1 \sqcap C_2 \mid & \text{- \textit{conjunction}} \\ & \forall R.C_1 \ . & \text{- \textit{(universal) value restriction}} \end{array} \tag{3.7}$$

where $A \in \text{CN}$ is a *concept name*, $R \in \text{RN}$ is a *role name*, and C_1, $C_2 \in \mathcal{ALC}$ are already constructed *(general) concepts*. Often, $\mathcal{ALC}$ is formulated with additional constructors: *top concept* $\top$, *disjunction* $C_1 \sqcup C_2$ and *existential restriction* $\exists R.C_1$, which are *dual* to the bottom concept, conjunction and value restriction respectively. We prefer to treat these constructors as abbreviations for $\neg\bot$, $\neg(\neg C_1 \sqcap \neg C_2)$ and $\neg(\forall R.\neg C_1)$ respectively.

Definition 3.2.3 (Semantics of $\mathcal{ALC}$)**.** The semantics of $\mathcal{ALC}$ is defined by means of interpretations. An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is a pair consisting of a non-empty set $\Delta^{\mathcal{I}}$ called the *domain* of interpretation and a mapping $\cdot^{\mathcal{I}}$ that assigns to every *concept name* $A \in \text{CN}$, a subset $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, and to every *role name* $R \in \text{RN}$, a binary relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. This assignment is extended inductively over definition (3.7) to *general concepts* as follows:

$$\begin{array}{rrcll} C^{\mathcal{I}} := & A^{\mathcal{I}} & = & \textit{given} & \mid \\ & \bot^{\mathcal{I}} & = & \{\} & \mid \\ & (\neg C_1)^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \setminus C_1{}^{\mathcal{I}} & \mid \\ & (C_1 \sqcap C_2)^{\mathcal{I}} & = & C_1{}^{\mathcal{I}} \cap C_2{}^{\mathcal{I}} & \mid \\ & (\forall R.C_1)^{\mathcal{I}} & = & \{a \in \Delta^{\mathcal{I}} \mid (\forall b.(a,b) \in R^{\mathcal{I}}) \Rightarrow b \in C_1{}^{\mathcal{I}}\} & . \end{array}$$

There are several reasoning problems for description logics, most of them have been discussed in chapter 2. We say that a concept C is *satisfiable* if there exists an interpretation $\mathcal{I}$ such that $C^{\mathcal{I}} \neq \{\}$. We say that a concept C_1 *subsumes* a concept C_2 (notation $C_2 \sqsubseteq C_1$) if for every interpretation $\mathcal{I}$, we have $C_2{}^{\mathcal{I}} \subseteq C_1{}^{\mathcal{I}}$. Concepts C_1 and C_2 are *equivalent* ($C_1 \doteq C_2$) if $C_1 \sqsubseteq C_2$ and $C_2 \sqsubseteq C_1$. Finally, we say that

[4] Although this has been recently reconsidered, see discussion in chapter 2

concepts C_1 and C_2 are *disjoint* ($C_1 \sqcap C_2 \doteq \bot$) if for every interpretation $\mathcal{I}$, we have ${C_1}^{\mathcal{I}} \cap {C_2}^{\mathcal{I}} = \{\}$. The *reasoning problem* is then: *given some input concepts, check whether they enjoy the respective property?* ◈

It is easy to see that for $\mathcal{ALC}$ all reasoning problems can be reduced to *concept satisfiability*. Indeed, it is easy to show that $C_2 \sqsubseteq C_1$ *iff* $C_2 \sqcap \neg C_1$ is unsatisfiable. Similarly, $C_1 \sqcap C_2 \doteq \bot$ if and only if concept $C_1 \sqcap C_2$ is unsatisfiable [Schmidt-Schauß & Smolka, 1991].

It was observed by Schild [1991] that $\mathcal{ALC}$ is a syntactical variant of the multi-modal logic $\mathbf{K_n}$ considered in the previous section, where modal formulas $\Box_i A$ correspond to universal value restrictions $\forall R_i.A$. Hence, one can mirror the relational translation (3.5) defined for modal formulas to $\mathcal{ALC}$ concepts as follows:

$$\begin{array}{rcrcll} \tau(C,x) & := & \tau(A,x) & = & A(x) & | \\ & & \tau(\bot,x) & = & \bot & | \\ & & \tau(\neg C_1,x) & = & \neg\tau(C_1,x) & | \\ & & \tau(C_1 \sqcap C_2,x) & = & \tau(C_1,x) \wedge \tau(C_2,x) & | \\ & & \tau(\forall S.C_1,x) & = & \forall y.[\tau(S,x,y) \rightarrow \tau(C_1,y)] & . \\ \tau(S,x,y) & := & \tau(R,x,y) & = & R(x,y)\ . & \end{array} \tag{3.8}$$

This translation maps $\mathcal{ALC}$-concepts to the following fragment of first-order logic:

$$\mathcal{FO}(\mathcal{ALC}) \ ::= \ A(x) \mid \bot \mid \neg F_1 \mid F_1 \wedge F_2 \mid \forall y.[R(x,y) \rightarrow F_1[y]]\ . \tag{3.9}$$

where $A(x)$ is a unary atom, $R(x,y)$ is a binary atom and F_1, $F_2 \in \mathcal{FO}(\mathcal{ALC})$.

It is easy to show that the first-order translation (3.8) *preserves satisfiability*, i.e., a concept C is satisfiable (according to Definition 3.2.3) if and only if its translation $\tau(C,x)$ is satisfiable (as a first-order formula). This correspondence can be used to solve reasoning problems for $\mathcal{ALC}$ through the first-order logic, i.e., using an algorithm that decides satisfiability of fragment (3.9). In chapter 4 we demonstrate how to construct such decision procedures.

One is usually interested not in solving a reasoning problem alone, but with respect to some *background terminology*. A *terminology* (or short TBox) is a collection of *(general) concept inclusions axioms* (short GCIs) of form $C_1 \sqsubseteq C_2$ and possibly some *role inclusion axioms* (RIAs) of form $R_1 \sqsubseteq R_2$ also called *role hierarchies*. We say that an interpretation $\mathcal{I}$ is a *model* of TBox if for every concept inclusion axiom $(C_1 \sqsubseteq C_2) \in$ TBox and for every role inclusion axiom $(R_1 \sqsubseteq R_2) \in$ TBox, we have ${C_1}^{\mathcal{I}} \subseteq {C_2}^{\mathcal{I}}$ and ${R_1}^{\mathcal{I}} \subseteq {R_2}^{\mathcal{I}}$. A concept C_1 is *satisfiable* w.r.t. a TBox, if ${C_1}^{\mathcal{I}} \neq \{\}$ for some model $\mathcal{I}$ of TBox. *Concept subsumption* and *concept disjointness* w.r.t. to a

TBox are defined similarly, and can be reduced to the *concept satisfiability* problem in the case of description logic $\mathcal{ALC}$.

From the first-order point of view, inclusion axioms correspond to implications between formulas constructed for concepts and roles: $C_1 \sqsubseteq C_2$ is translated to $\forall x.[\tau(C_1, x) \rightarrow \tau(C_2, x)]$ and $R_1 \sqsubseteq R_2$ is translated to $\forall xy.[\tau(R_1, x, y) \rightarrow \tau(R_2, x, y)]$. Hence, in order to check satisfiability of a concept C w.r.t. a TBox, one has to check satisfiability of the following first-order formula:

$$\bigwedge_{(C_1 \sqsubseteq C_2) \in \text{TBox}} \forall x.[\tau(C_1, x) \rightarrow \tau(C_2, x)] \quad \wedge \\ \wedge \bigwedge_{(R_1 \sqsubseteq R_2) \in \text{TBox}} \forall x.[\tau(R_1, x, y) \rightarrow \tau(R_2, x, y)] \quad \wedge \quad \tau(C, x) \tag{3.10}$$

Since we have introduced a *basic* description logic, there must be some other extensions. Actually, we have already described an extensions of $\mathcal{ALC}$ with *role hierarchies*, which is usually denoted by $\mathcal{ALCH}$. Other extensions are obtained by adding additional *constructors* for concepts and roles. In Table 3.2 we sketch some well-known constructors for description logics, and provide their first-order translation. We distinguish DL-constructors of three types: (**1**) *role constructors* (**2**) *concept constructors* and (**3**) *TBox-declarations*.

Table 3.2 Some constructors for description logics

Constructor	$\mathcal{FO}$ *− translation*	*- Name*	*Not.*
S^-	$\tau(S, y, x)$	- *inverse roles*	$\mathcal{I}, (^-)$
$S \sqcap T$	$\tau(S, x, y) \wedge \tau(T, x, y)$	- *conjunction of roles*	$(\sqcap)$
$\neg S$	$\neg\tau(S, x, y)$	- *negation of roles*	$(\neg)$
$S \circ T$	$\exists z.[\tau(S, x, z) \wedge \tau(T, z, y)]$	- *composition of roles*	$(\circ)$
$(\leqslant n\, S)$	$\forall y_1 ... y_n.[\ (\bigwedge_{1 \leq i \leq n} \tau(R, x, y_i)\ ) \rightarrow \bigvee_{1 \leq i < j \leq n} (y_i \simeq y_j)]$	- *number restrictions*	$\mathcal{N}$
$(\leqslant n\, S.C)$	$\forall y_1 ... y_n.[\ (\bigwedge_{1 \leq i \leq n} \tau(R, x, y_i) \wedge \tau(C, y_i)\ ) \rightarrow \bigvee_{1 \leq i < j \leq n} (y_i \simeq y_j)]$	- *qualified num. restr.*	$\mathcal{Q}$
$S \subseteq T$	$\forall y.[\tau(S, x, y) \rightarrow \tau(T, x, y)]$	- *role-value maps*	$(\subseteq)$
$S \sqsubseteq T$	$\forall xy.[\tau(S, x, y) \rightarrow \tau(T, x, y)]$	- *role hierarchies*	$\mathcal{H}$
Transitive(T)	$\forall xyz.[\tau(T, x, y) \wedge \tau(T, y, z) \rightarrow \tau(T, x, z)]$	- *transitive roles*	$\mathcal{S}^a$
Functional(S)	$\forall xyz.[\tau(S, x, y) \wedge \tau(S, x, z) \rightarrow (y \simeq z)]$	- *functional roles*	$\mathcal{F}$
Nominal(C)	$\exists x.[\tau(C, x)] \wedge \forall xy.[\tau(C, x) \wedge \tau(C, y) \rightarrow (x \simeq y)]$	- *nominals*	$\mathcal{O}$

[a] An extension of $\mathcal{ALC}$ with transitive roles is denoted by $\mathcal{S}$

Role constructors: For $\mathcal{ALC}$ we did not allow construction of compound roles. In the first part of Table 3.2, we introduce additional operations that allow to construct new *role expressions* (we denote them by S or T), from *atomic roles* $R \in \mathrm{RN}$. Since a role S intuitively corresponds to a *binary* relation $S(x, y)$, the simplest operation would be to *inverse* this relation by swapping its arguments: $S^{\smile}(x, y) := S(y, x)$. This gives rise to so-called *inverse roles*: S^{-} is the *inverse* of role S. By analogy to construction of concepts, one can introduce *Boolean combinations of roles*: *conjunction of roles* $S \sqcap T$ and *negation of roles* $\neg S$, which also make it possible to express a *disjunction of roles* $S \sqcup T$ as $\neg(\neg S \sqcap \neg T)$. The last role constructor that we introduce here is *composition of roles* $S \circ T$ which allows to *chain* (or *compose*) two or more roles. If we look at the first-order translation for role compositions given in Table 3.2 this would probably remind us the first-order translation for *existential restrictions* $\exists S.C$ (by analogy, $S \circ T$ must probably be written as $\exists S.T$).

Concept constructors: Additional concept constructors that we consider, express so-called *counting* in description logics. First constructor $({\leqslant}\, n\, S)$ called *at most number restriction* represents a set of elements that are related to at most n other elements by means of role S. The dual *at least number restriction* $({\geqslant}\, n\, S)$ is an abbreviation for $\neg({\leqslant}\, (n-1)\, S)$.[5] These *number restrictions* can be generalised to *qualified number restrictions* of form $({\leqslant}\, n\, S.C)$ and $({\geqslant}\, n\, S.C)$, denoting the sets of elements that are S-connected to at most, respectively at least, n elements *from* C. Note that $({\leqslant}\, n\, S)$ and $({\geqslant}\, n\, S)$ are equivalent to $({\leqslant}\, n\, S.\top)$ and $({\geqslant}\, n\, S.\top)$ respectively. The last concept constructor we introduce here is *role-value map*: $S \subseteq T$ represents a set of elements x such that every element y that is S-connected with x, is also T-connected with x. Note that in presence of Boolean role operations, this constructor is equivalent to $\forall(S \sqcap \neg T).\bot$.

TBox-declarations: In the lower part of Table 3.2 we have listed some TBox-declarations. We have already discussed *role inclusion axioms* (*role hierarchies*) $S \sqsubseteq T$. Note the difference between this declaration and a *role-value map* $S \subseteq T$. The last describes a concept, whereas the first expresses that role S is a *subrole* of T. For example, it is easy to show that $(S \subseteq T) \doteq \top$ w.r.t. every TBox containing $S \sqsubseteq T$. Declarations $\mathsf{Transitive}(T)$ and $\mathsf{Functional}(S)$ restrict role expressions T and S to be interpreted by transitive and functional relations respectively. $\mathsf{Nominal}(C)$ expresses that concept C must be interpreted with a one-element set. In chapter 2 we have used another notation for nominals: expression $\{a\}$ denotes a nominal formed from an *individual* a. These notations are essentially equivalent: declaration

[5] Here and everywhere else expressions n and $(n-1)$ in number restrictions are fixed positive integers and *not* arithmetical expressions

$\mathsf{Nominal}(C)$ is equivalent to $C \doteq \{a\}$, where a is a fresh individual for C.

Different description logics combine different sets of the constructors which is indicated by the correspondent letters given in the last column of Table 3.2. For example, description logic $\mathcal{ALC}(^-)$ also denoted by $\mathcal{ALCI}$ is an extension of the basic description logic with *inverse roles*. Similarly, $\mathcal{ALC}(^-, \sqcap, \neg)$ is the extension of $\mathcal{ALCI}$ with the Boolean combinations of roles. A very expressive description logic $\mathcal{SHIQ}$ [see Horrocks et al., 2000], is an extension of $\mathcal{ALC}$ with *transitive roles* ($\mathcal{S}$), *role hierarchies* ($\mathcal{H}$), *inverse roles* ($\mathcal{I}$) and *qualified number restrictions* ($\mathcal{Q}$)[6]. Description logic $\mathcal{SHOIN}$ is an extension of $\mathcal{SHI}$ with *nominals* ($\mathcal{O}$) and *number restrictions* ($\mathcal{N}$), which corresponds to the ontology language for the Semantic Web OWL DL [Horrocks & Patel-Schneider, 2004].

Not every combination of constructors listed in Table 3.2 results in a description logic with decidable subsumption problem. It is well-known that $\mathcal{SHIQ}$ becomes *undecidable* when *non-simple roles* are allowed in number restrictions [see Horrocks et al., 2000]. A role is *simple* if it does not contain a *transitive subrole* in w.r.t. a role hierarchy. Another well-known example of undecidable description logic, is an extension of $\mathcal{ALC}$ with *composition of roles* and *role-value maps* $\mathcal{ALC}(\circ, \subseteq)$. In fact even much simpler description logics from KL-ONE-family [Brachman & Schmolze, 1985] become undecidable [see Schmidt-Schauß, 1989] when composition of roles is allowed (see also section 2.5 of this thesis and [Donini, 2003]).

We see that extensions of description logics with transitive roles and role compositions in many cases lead to undecidability of the subsumption problem. However, *compositional axioms* are often indispensable for conceptual modelling, in particular for development of medical terminologies [see Spackman, 2000; Rector, 2002]. There are not many know *decidable* description logics known that can admit role compositions in some form. A notable exception is a description logic $\mathcal{EL}$ and its extensions, which were discussed in chapter 2 and a recent result by Horrocks & Sattler [2004], describing a tableau-based procedure for $\mathcal{SHIQ}$ with so-called *acyclic* compositional axioms. In this thesis we try to understand this problem from first-order point of view using the translations given above, and propose some new solutions.

3.2.3 Reasoning in Modal and Description Logics

The *possible world semantics* for modal logics (see Definition 3.2.1) led to the development of so-called *tableau-based procedures*. Given a modal formula F, a procedure tries to construct a *Kripke model* for F. It starts with the initial wold w_0 in which the formula F must be satisfied (notation $w_0 : F$) and applies an *expansion rule*

[6] although, see a restriction below

corresponding to the top symbol of the formula. In System 1 we give a collection of

$$\alpha : \frac{w : F_1 \wedge F_2}{w : F_1 \quad w : F_2} \qquad \beta : \frac{w : \neg(F_1 \wedge F_2)}{w : \neg F_1 \parallel w : \neg F_2} \qquad \mathtt{K} : \frac{w : \Box F \quad wRw'}{w' : F} \qquad \pi : \frac{w : \neg\Box F}{wRw' \quad w' : \neg F}$$

$$\Big[\textit{where } w' \textit{ is fresh} \Big]$$

$$\mathtt{dneg} : \frac{w : \neg\neg F}{w : F} \qquad \mathtt{clash} : \frac{w : F \quad w : \neg F}{\bot} \qquad \Big| \quad 4 : \frac{w : \Box F \quad wRw'}{w' : \Box F}$$

System 1: Tableau expansion rules for MLs **K** and **K4**

expansion rules for modal logics **K** and **K4**. The tableau system for **K** consists of rules α, β, `K`, π, `dneg` and `clash`. The tableau system for **K4** is obtained by adding rule `4`. Rule β is *nondeterministic* and causes branching of the inference procedure. An input formula is satisfiable *iff* there exists a branch which is closed under all inferences and does not contain the contradiction $\bot$. See e.g., [Massacci, 2000] for descriptions of other tableau-based systems for modal logics.

The development of reasoning procedures for description logics has been affected by the fact that the first description logics, like Kl-One [Brachman & Schmolze, 1985] have been too expressive, and the reasoning problems for them were in general undecidable [Schmidt-Schauß, 1989]. Hence, the first procedures for reasoning in such languages were *incomplete*. For example, subsumption of concepts have been mainly solved using so-called *structural subsumption algorithms* which were complete only for very restrictive languages without disjunction and full negation.

The first step towards the development of reasoning procedures for *expressive* terminological languages was made in [Schmidt-Schauß & Smolka, 1991], where a sound and *complete* reasoning procedure for DL $\mathcal{ALC}$ has been found. This was the first reasoning procedure to handle concept descriptions involving full negation and disjunctions of concepts. As it turned out later, Schmidt-Schauß & Smolka [1991] have essentially rediscovered the tableau procedures known for the modal logics – Schild [1991] was the first to notice that $\mathcal{ALC}$ is merely a syntactical variant of well-known multi-modal logic $\mathbf{K_n}$. Since then, the tableau procedures for modal logics have been transferred to their respective description logics, and extended to handle additional constructors, like (qualified) number restrictions or nominals.

In System 2 we gave a tableau calculus for checking satisfiability of $\mathcal{ALC}$-concepts in the form it is usually presented today[7] (recall that other reasoning problems for $\mathcal{ALC}$ can be reduced to concept satisfiability). Here it is assumed that the initial

[7] The original algorithm by Schmidt-Schauß & Smolka [1991] used essentially the same rules, but they were applied to a collection of so-called *constraints* that were generated for the input concept, rather than to the concept directly

$$\sqcap : \frac{w : C_1 \sqcap C_2}{w : C_1 \quad w : C_2} \qquad \sqcup : \frac{w : C_1 \sqcup C_2}{w : C_1 \parallel w : C_2} \qquad \forall : \frac{w : \forall R.C \quad wRw'}{w' : C} \qquad \exists : \frac{w : \exists R.C}{wRw' \quad w' : C}\ \left[\textit{where } w' \textit{ is fresh}\right]$$

$$\bot : \frac{w : A \quad w : \neg A}{\bot}$$

System 2: Tableau expansion rules for DL $\mathcal{ALC}$

concept is first converted into a *negation normal form*, e.g., negation is applied to atomic concepts only. This basic tableau procedure has been extended to many expressive description logics, including those mentioned in subsection 3.2.2. Despite the fact that the tableau-based procedures have a relatively high complexity (which is sometimes much worse than the optimal one), many sophisticated optimisation techniques have been developed which made these procedures usable in practice [see Horrocks et al., 2000]. In fact, the most efficient DL-reasoners today, like Fact [Horrocks, 1998] and Racer [Haarslev & Möller, 2001], employ tableau-based procedures.

Despite a number of positive sides, tableau-based procedures have some drawbacks. First, such procedures require the underlying logic to possess a some form of the *tree model property*, since such models are constructed in the expansion. Second, the tableau procedures for expressive languages are rather *ad-hoc*, and it is often not easy to justify their *correctness*. To demonstrate the second point, take a look at the expansion rule 4 from System 1. This rule corresponds to the transitivity condition on relation R, however there is no obvious connection between this rule and the transitivity axiom. Moreover, unrestricted application of such rule may cause a tableau procedure to loop, and a special loop-detection mechanism must be provided to ensure the termination. These drawbacks could be a serious obstacle towards development of reasoning procedures for new *ontology languages* which go beyond the description logics studied so far, like the W3C-recommended ontology language for the Semantic Web OWL [see Horrocks & Patel-Schneider, 2004].

3.3 Decidable Fragments of First-Order Logic

3.3.1 Prefix-vocabulary Classes

A "hunt" for *decidable fragments* of first-order logic, began shortly after Church and Turing in 1930's discovered undecidability of the full first-order logic. To compensate this negative result, logicians aimed to identify and classify decidable and undecidable classes of first-order formulas. In this quest, they were focused initially on so-called *prefix-vocabulary classes* – sets of first-order formulas in prenex normal form with a given quantifier pattern and a given vocabulary of predicate and functional symbols. After almost fifty years of research, such *complete* classification has been found, and for most fragments optimal complexity results were obtained.

This section does *not* aim to (and actually cannot) give an overview of these and related results, for which the reader is forwarded to the book [Börger, Grädel & Gurevich, 1997]. In this section we sketch two particular first-order classes that will be of some interest in this thesis.

The monadic classes

The *monadic class* of first-order logic, also called the *Löwenheim class*, is a set of first-order formulas without functional symbols and containing only unary (*monadic*) predicate symbols. Decidability of this class is known since [Löwenheim, 1915]. In abstract grammar notation, we define this class by:

$$\mathcal{M} \ ::= \ A[x] \mid \neg M_1 \mid M_1 \wedge M_2 \mid \forall x.M_1 \ . \tag{3.11}$$

where $A[x]$ is a unary atom and M_1, $M_2 \in \mathcal{M}$.

The monadic class is known to be decidable even if equality is allowed. In this case we deal with the *monadic class with equality*:

$$\mathcal{M}_{\simeq} \ ::= \ A[x] \mid (x \simeq y) \mid \neg M_1 \mid M_1 \wedge M_2 \mid \forall x.M_1 \ . \tag{3.12}$$

The *monadic fragment without equality* can be also extended with unary functional symbols to a so-called *full monadic class* (also known as *Löb-Gurevich class*):

$$\mathcal{M}_f \ ::= \ A[x] \mid M_1[x] \cdot \{x/f(x)\} \mid \neg M_1 \mid M_1 \wedge M_2 \mid \forall x.M_1 \ . \tag{3.13}$$

where $f(x)$ is a unary functional symbol. The second case means that a new monadic formula can be formed from a monadic formula with one variable by replacing this variable with a unary function. For example, $\exists \mathtt{y}.(\mathtt{b}(\mathtt{y}) \wedge \forall \mathtt{z}.[\mathtt{a}(\mathtt{z}) \vee \mathtt{b}(\mathtt{f}(\mathtt{y}))] \in \mathcal{M}_f$, where existential quantifier and disjunction are expressible by means of other constructors as usual.

The *monadic fragment with equality* and the *full monadic fragment* are two *maximal* decidable fragment of first-order logic of complexity NTIME($2^{O(n/\log n)}$) and NTIME($2^{O(n)}$) respectively. For the first fragment a matching lower bound has been found (even without equality and with restricted quantification) [see Börger et al., 1997].

An interest to monadic fragments has been renewed in [Bachmair, Ganzinger & Waldmann, 1993*a*] where it was noticed that *set constraints* studied in program analysis, correspond to the monadic class.

The Gödel and Goldfarb classes

Another well-known decidable class of first-order formulas is a so-called *Gödel class*, which is a set of first-order sentences of form:

$$\exists x_1. \ldots \exists x_k . \forall y_1 . \forall y_2 . \exists x_{k+1} . \ldots \exists x_n . F \tag{3.14}$$

where F is a quantifier-free formula. It is usually said that such formulas have *quantifier prefix* $\exists^* \forall^2 \exists^*$. Gödel [1933*b*] proved decidability for the set of sentences of form (3.14) by establishing a *finite model property*: every satisfiable formula of this form must have a finite model. Later, an exact bound on the size of finite models has been found, which implies that the Gödel fragment is NEXPTIME-complete [see Börger et al., 1997].

All the above results were proven for the case when formulas of form (3.14) do not contain equality. In the original paper Gödel [1933*b*] has conjectured that his results hold also for the case with equality. Only fifty years later, Goldfarb [1984] has demonstrated that this conjecture is actually not true. In fact a *Goldfarb class*, which is a set of sentences possibly with equality over a quantifier prefix $\forall\forall\exists$, is already undecidable [see Börger et al., 1997].

3.3.2 Two-Variable Fragments

Some classes of formulas do not really fit to prefix-vocabulary classes, for example, the modal fragment (3.6) obtained by translation (3.5) of modal formulas described in subsection 3.2.1. Indeed, it is easy to see that a direct transformation of formulas from (3.5) into prenex normal form may result in arbitrary quantifier prefix. Hence, it seems that prefix vocabulary classes cannot help explaining decidability of modal logics.

If we examine fragment (3.6) more closely, we noticed that every subformula of formulas constructed according to this definition contains at most two free variables, in other terms the *width* of these formulas is bounded by two. An intuition may suggest us to consider a set of first-order formulas with such property. This class of formulas is called the *two-variable fragment* and can be recursively defined using the following grammar:

$$\mathcal{FO}^2 \;::=\; A[x,y] \mid \neg T_1 \mid T_1[x,y] \wedge T_2[x,y] \mid \forall y . T_1[x,y] \; . \tag{3.15}$$

where A is an atom (possibly equality) and T_1, $T_2 \in \mathcal{FO}^2$ (here notation $F[x_1,..,x_n]$ means that *free*$[F] \in \{x_1,..,x_n\}$). It is possible to generalise the two-variable fragment to so-called *bounded-variable fragments*: $\mathcal{FO}^k$ denotes the set of formulas whose *width* is bounded by k: $\mathcal{FO}^k := \{F \in \mathcal{FO} \mid \mathit{width}(F) \leq k\}$. Fragment $\mathcal{FO}^k$ can be also seen as sets of formulas that can be constructed using *k variable names*.

A relationship between modal logics and the two-variable fragment has been already observed by Gabbay [1981] who tried to explain the good computational properties of modal logic using the first-order logic (see also [Ohlbach et al., 2001] for a related discussion).

Scott [1962] described a translation that maps every two-variable formula to an *equisatisfiable* formula of form:

$$\forall xy.F[x,y] \;\wedge \bigwedge_{1\leq i\leq n} \forall x.\exists y.G_i[x,y] \tag{3.16}$$

where F and G_i, $1 \leq i \leq n$ are quantifier-free formulas. This form of two-variable formulas is called the *Scott normal form*.[8] It is easy to see that formulas of form (3.16) belong to the *Gödel class*. Hence, Scott translation provides an elegant reduction from the satisfiability problem for two-variable formulas to that of the *Gödel class*. However, as the *Gödel class with equality* was proven to be undecidable, the Scott reduction has guaranteed only decidability for the two-variable fragment *without equality*. Decidability of the full two-variable fragment has been established later by Mortimer [1975] who showed that this fragment has a finite model property. Grädel, Kolaitis & Vardi [1997] have improved the result of [Mortimer, 1975] by showing that every two-variable formula (with equality) has a model of size $2^{O(n)}$, and demonstrated that $\mathcal{FO}^2$ is NEXPTIME-complete (see also [Börger et al., 1997, Section 8.1]).

Many extensions of two-variable fragment have been considered. However, most of them appear to be undecidable, in particular, an extension of two-variable fragment with equivalence relations (see [Grädel & Otto, 1999] for an overview). A notable exception is the *two-variable fragment with counting* $\mathcal{C}^2$. This fragment extends $\mathcal{FO}^2$ with *counting quantifiers* of form $\exists^{\leq n}y.F$ and $\exists^{\geq n}y.F$, which mirror counting restrictions in modal and description logics. Decidability of this extension has been shown by a very complicated model construction [Grädel, Otto & Rosen, 1997; Pacholski, Szwast & Tendera, 2000].

3.3.3 Guarded Fragments

From the previous section, we can conclude that, although the two-variable fragment can explain decidability of modal-like languages, it does not really explain their moderate *complexity* and decidability of its *extensions*. Indeed, as has been noted, two-variable fragment is NEXPTIME-complete, whereas most modal logics are merely in PSPACE.

[8] In subsection 3.5.7 we introduce a *structural transformation* which generalises the Scott reduction

Trying to explain this phenomenon, we take an even closer look at formulas in (3.6) of the *modal fragment* of first-order logic $\mathcal{FO}(\mathrm{MF})$. We may notice that every quantification in these formulas is *bounded*, i.e., the range of quantified variables is limited by an atom-*guard*, in that case $R(x, y)$. Motivated by this observation, Andréka et al. [1996] introduce the *guarded fragment* of first-order logic:

$$\mathcal{GF} \ ::= \ A \mid \neg F_1 \mid F_1 \wedge F_2 \mid \forall \overline{y}.[G \rightarrow F_1] \ . \tag{3.17}$$

where A is an atom (possibly equality), F_1, $F_2 \in \mathcal{GF}$, $\overline{y}$ is some vector of variables y_1,.., y_n (so, $\forall \overline{y}.F$ is a shortcut for $\forall y_1. \cdots \forall y_n.F$), and G in formulas of form $\forall \overline{y}.[G \rightarrow F_1]$ is an atom called *guard* that contains all free variables of F_1 , i.e., in symbols, $\mathit{vars}[F_1] \subseteq \mathit{vars}[G]$. Note that the formulas of the guarded fragment are no longer restricted to two variables.

We also admit the *guarded existential quantification* in guarded formulas of form $\exists \overline{y}.[G \wedge F_1]$, which we view as a shortcut for $\neg(\forall \overline{y}.[G \rightarrow \neg F_1])$. According to definition (3.17) and our convention, formula $\forall \mathtt{x}.(\underline{\mathtt{V(x)}} \rightarrow \exists \mathtt{y}.[\underline{\mathtt{E(x,y)}} \wedge \mathtt{V(y)}])$ expressing seriality of a graph is guarded (the appropriate guards are underlined), whereas a *transitivity axiom*: $\forall \mathtt{xyz}.[\underline{\mathtt{T(x,y)}} \wedge \underline{\mathtt{T(y,z)}} \rightarrow \mathtt{T(x,z)}]$ is not, since both candidates for a guard do not contain all variables of the remaining atoms (in fact, it can be shown that no guarded formula can express transitivity).

Andréka et al. [1996] and Grädel [1999] have noticed that the *guarded fragment* inherits many nice computational properties from modal logics, the most important from which is the *tree-model property*. Vardi [1996] gives strong arguments that the *tree-model property* is a key property which makes modal logics so *robustly decidable*. And indeed, the *guarded fragment* has much better computational properties than the *two-variable fragment*: although the full guarded fragment has been shown to be 2EXPTIME-complete, its *bounded variable* part $\mathcal{GF}^k := \mathcal{GF} \cap \mathcal{FO}^k$ is "only" EXPTIME-complete (see [Grädel, 1999]). The guarded fragment behaves robustly w.r.t. some extensions that were undecidable with two-variable fragments, notably, with fixed-point constructors [Grädel & Walukiewicz, 1999].

However, there is still a gap between modal logics and the guarded fragment, both w.r.t. complexity and robust decidability. It has been shown that extensions of the guarded fragment with functional binary relations (which is a very restricted form of counting) and transitive relations, are undecidable [Grädel, 1999]. For transitivity, the situation does not improve even when restricting to two-variable guarded fragment: $\mathcal{GF}^2$ [Ganzinger et al., 1999][9]. Even when transitive relations are restricted to be guards only, $\mathcal{GF}^2$ is 2EXPTIME-complete [Szwast & Tendera, 2001; Kieronski, 2003].

[9] In this thesis we demonstrate that even two transitive relations suffice for this

3.4 Domino Problems and Undecidability

Reduction from *domino* (also called *tiling*) *problems* is nowadays the most commonly used method for proving of undecidability results.

Domino problems have been introduced by Wang [1961][10] as a tool for proving undecidability of some prefix-vocabulary classes. Several variations of domino problems have been proposed, which became a convenient tool for deriving the lower complexity bounds for different problems (see [Börger et al., 1997] for a history and an overview of domino problems). In this thesis we obtain new undecidability results for certain fragments of first-order logic by a reduction from a most commonly used variant of domino problems:

Definition 3.4.1 (Domino Problem)**.** A *domino system* $\mathcal{D}$ is a triple (D, H, V), where D is a finite set of *dominoes* and H, $V \subseteq D \times D$ are two binary relations. A *tiling* of a *grid* $\mathbb{N} \times \mathbb{N}$ for $\mathcal{D}$ is an assignment $\tau : \mathbb{N} \times \mathbb{N} \to D$ such that for every $i, j \in \mathbb{N}$ the following conditions hold: (***i***) $(\tau(i,j), \tau(i+1,j)) \in V$ and (***ii***) $(\tau(i,j), \tau(i,j+1)) \in H$.

A *domino problem* is, given a domino system $\mathcal{D} = (D, H, V)$, check whether there exists a tiling for $\mathcal{D}$ of a grid $\mathbb{N} \times \mathbb{N}$? ◈

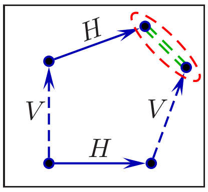
The domino problem formulated in Definition 3.4.1 is undecidable. For reducing a problem to the domino problem, one usually needs to perform the following steps: (**1**) encode the tiling conditions of a domino system, and (**2**) encode a *grid* structure.

The first step is usually easy. For example, let $\mathcal{G} = \mathbb{N} \times \mathbb{N}$ be a grid, $H(x, y)$, $V(x, y)$ be the horizontal and vertical neighbouring relations in $\mathcal{G}$ and $d_i(x)$ be a unary predicate that represents the sets of nodes from $\mathcal{G}$ to which a tile $d_i \in D$ is assigned, $1 \leq i \leq |D|$. Then the tiling conditions from Definition 3.4.1 can be encoded using a first-order sentence TILING defined in Figure 3.1. Note that $\mathsf{TILING} \in \mathcal{GF}^2$, so it belongs to both the *two-variable fragment* and the *guarded fragment* of first-order logic.

Figure 3.1 Encoding of tiling conditions in the first-order logic

$$
\begin{array}{rlll}
\mathsf{TILING} & := \displaystyle\bigwedge_{1 \leq i < j \leq |D|} & \forall x.(\, d_i(x) \to [\neg d_j(x)]\,)\ \wedge & \text{- } \textit{assignment } \tau : \mathcal{G} \to D \textit{ is functional} \\
\wedge\ \forall xy.(\, H(x,y) \to & \displaystyle\bigvee_{(d_i, d_j) \in H} & [d_i(x) \wedge d_j(y)]\,)\ \wedge & \text{- } \textit{condition } (i) \textit{ of Definition 3.4.1} \\
\wedge\ \forall xy.(\, V(x,y) \to & \displaystyle\bigvee_{(d_i, d_j) \in V} & [d_i(x) \wedge d_j(y)]\,) & \text{- } \textit{condition } (ii) \textit{ of Definition 3.4.1}
\end{array}
$$

[10] Tiling problems are closely related to two-counter Minsky machines [Minksy, 1961]



The hardest part in reduction from a domino problem is to enforce an infinite grid structure for the set $\mathcal{G}$. One usually needs to encode the following *confluence property* for relations H and V: whenever we have $H(x,y)$, $V(x,z)$, then there should be a point u such that $V(y,u)$ and $H(z,u)$ hold. Using inverse and composition, this property can be shortly written as: $H^{\smile} \circ V \subseteq V \circ H^{\smile}$. In many reduction proofs, one often first enforces a *tree structure* for relations H and V and then "glues" the appropriate nodes of this tree to obtain a grid (say by additional constraints with equality).

There are many other variants of domino problems that are either known to be undecidable, or complete for certain complexity classes [see Börger et al., 1997]. We just briefly mention a *periodic tiling* problem. A domino system $\mathcal{D} = (D, H, V)$ admits *periodic tiling* if there exists a tiling $\tau : \mathbb{N} \times \mathbb{N} \to D$, which satisfies all conditions from Definition 3.4.1, and additionally condition (***iii***): there exist naturals $n > 0$, $m > 0$ such that for all $i \geq 0$, $j \geq 0$, we have $\tau(i{+}n, j) = \tau(i, j) = \tau(i, j{+}m)$. The *periodic tiling problem* for domino systems is *undecidable* as well. In fact sets of domino systems that admit, respectively, periodic tiling, and no tiling are *recursively inseparable*, i.e., there is no decidable set of domino problems which contain the first sets but does not intersect with the second. By expressing the grid and the tiling conditions within some class of formulas, one usually shows that this class forms a *conservative reduction class*, i.e., the set of its formulas that admit a finite, and respectively, no model, are *recursively inseparable* [see Börger et al., 1997, p.90].

3.5 A Framework of Saturation-Based Theorem Proving

In this section we introduce a framework of saturation-based theorem proving in its modern form due to Bachmair & Ganzinger [1990, 1994]. We formulate several calculi used in automated deduction, that will be the basis of decision procedures that we describe afterwards. A more detailed overview for the material in this section can be found in [Bachmair & Ganzinger, 1998*a*, 2001; Nieuwenhuis & Rubio, 2001], and the technical part including all proofs can be found in [Kazakov, 2005].

3.5.1 Saturation-Based Theorem Proving

The goal for a *saturation-based theorem prover* is to establish *unsatisfiability* of an input clause set. In order to prove unsatisfiability, the prover applies inferences to clauses using a dedicated inference system called a *calculus*. The conclusion of every inference is added to the current clause set, and this process, called *saturation* is iterated until either a contradiction is derived, or the closure under all inferences is computed without deriving the contradiction. It is also possible that the process of saturation is continued without reaching a fixed-point.

In saturation-based theorem proving, every clause $C = L_1 \vee \cdots \vee L_k$ is treated as the multiset consisting of its literals $L = \{L_1, .., L_k\}_m$. In other words, the order of literals plays no rôle. A special *empty clause* $\square$ is introduced that corresponds to the empty multiset $\{\}_m$. This clause, which is always `false`, plays the rôle of a basic contradiction that one needs to derive in order to establish unsatisfiability of a clause set.

A calculus used in saturation-based theorem provers is usually given by a collection $\boldsymbol{\mathcal{S}}$ of conditional inference rules (schemes) of form:

Inference Rule

$$\mathtt{IR}: \frac{C_1, \ldots, C_n}{C} \tag{3.18}$$

$$\Big[\textit{where some conditions of the rule hold} \qquad\qquad \Big]$$

Here IR is a short identifier of the rule, $C_1, \ldots, C_n$ are the premises of the rule and C is its conclusion of the rule. The conditions written below the rule restrict applicability of the rule. We write $N \vdash_{\boldsymbol{\mathcal{S}}} C$ when the clause C is derivable from N using inferences from $\boldsymbol{\mathcal{S}}$. A calculus $\boldsymbol{\mathcal{S}}$ is *sound* if $N \nvdash_{\boldsymbol{\mathcal{S}}} \square$ for every satisfiable clause set N. It is *refutationally complete* if $N \vdash_{\boldsymbol{\mathcal{S}}} \square$ for every unsatisfiable clause set N. Soundness and completeness ensure that the calculus can be used correctly for checking unsatisfiability of clause sets.

Below we give calculi that are commonly used in saturation-based theorem provers. Most calculi allow for several optional parameters. These are typically defined by *selection strategies* and *ordering restrictions*. Such parameters can be used to influence, yet indirectly, the saturation process, which, in the end, makes it possible to design saturation-based decision procedures. We put a special effort to give a precise classification of admissible parameters for each calculus, so that flexible saturation strategies can be found afterwards. A detailed technical exposition of the material in this section can be found in [Kazakov, 2005].

3.5.2 The Ordered Resolution Calculus

Resolution calculus was invented by Robinson [1965] and later became the most successful method for automated reasoning in first-order logic. The *ordered resolution calculus* $\mathcal{OR}^{\succ}_{\mathit{Sel}}$ given in System 3, is the modern (refined) version of the Robinson's [1965] resolution calculus. This calculus is parametrised by an ordering $\succ$ on literals

Ordered Resolution

$$\mathtt{OR}: \frac{C \vee \underline{\boldsymbol{A}}^{\star} \quad D \vee \neg\underline{\boldsymbol{B}}}{C\sigma \vee D\sigma}$$

$\left[\textit{where (i) } \sigma = \mathrm{mgu}(A,B)\textit{; (ii) } A \textit{ is eligible strictly maximal w.r.t. } C \textit{ and } \sigma \textit{ and (iii) } \neg B \textit{ is eligible w.r.t. } D \textit{ and } \sigma.\right]$

Ordered Factoring

$$\mathtt{OF}: \frac{C \vee \underline{B} \vee \underline{\boldsymbol{A}}}{C\sigma \vee A\sigma}$$

$\left[\textit{where (i) } \sigma = \mathrm{mgu}(A,B)\textit{; (ii) } A \textit{ is eligible w.r.t. } C \textit{ and } \sigma.\right]$

System 3: The ordered resolution calculus with selection $\mathcal{OR}^{\succ}_{Sel}$

and a selection function *Sel* for negative literals which are used in conditions of the inference rules according to the following terminology:

A *selection function* is a mapping *Sel* that assigns to every clause C a (possibly empty) sub-multiset $Sel(C)$ of negative literals from C. These literals are then called the *selected literals* in C. A literal L is *maximal* w.r.t. C, if $L' \succ L$ for no literal $L' \in C$. Additionally, if $L \notin C$, then L is *strictly maximal w.r.t.* C. Given a clause C and a substitution σ, we say that a literal L *is eligible (strictly maximal) w.r.t.* C *and* σ if either $L \in Sel(C \vee L)$, or otherwise $Sel(C \vee L) = \{\}$ and $L\sigma$ is (strictly) maximal w.r.t. $C\sigma$.

The ordered resolution calculus $\mathcal{OR}^{\succ}_{Sel}$ is refutationally complete not for all orderings, but only for admissible ones. We say that an ordering $\succ$ is *liftable* if there exists an order $\succ_0$ on ground expressions such that $E_1 \succ E_2$ implies that $E_1\sigma \succeq_0 E_2\sigma$ for every ground substitution σ. Then admissible orderings are defined as follows:

Definition 3.5.1. An ordering $\succ$ on literals is *admissible for ordered resolution* if it admits the following conditions:

(**L**) $\succ$ is liftable;
(**W**) $\succ$ is a well-order on ground literals;
(**R1**) $\neg A \succ A$ for every ground atom A. $\Diamond$

Ordering restrictions and selection strategies are examples of so-called *local* refinements of saturation-based calculi. They allow one to reduce the number of inferences and thereby, to prune the search space of the prover. The efficiency of modern theorem provers, however, is highly determined by *global* simplification techniques of a prover, in which the notion of *redundancy* plays the key rôle.

Let $\succ$ be an ordering on clauses, that is the multiset extension of the ordering $\succ$ on literals. Let N be a set of ground clauses. A ground clause C is *redundant w.r.t.* N, if C follows logically from the set $N_C := \{C' \mid C' \prec C\}$ of the clauses from N that are smaller than C. A ground inference $\pi = (C_1, .., C_k \vdash C)$ with the maximal premise C_1 is *redundant w.r.t.* N if C follows logically from the set $N_{C_1} := \{C' \mid C' \prec C_1\}$.

A (possibly non-ground) clause C is *redundant w.r.t. a set* N of (possibly non-ground) clauses, if every ground instance $C^0 = C{\cdot}\sigma^0$ of C is redundant w.r.t. the set N^{gr} of ground instances of the clauses from N. Similarly, a (possibly non-ground) inference $\pi = (C_1, .., C_k \vdash C)$ is *redundant w.r.t.* N if every ground instance $\pi^0 = (C_1\sigma^0, .., C_k\sigma^0 \vdash C\sigma^0)$ of π is redundant w.r.t. N^{gr}.

We say that a clause set N is *saturated up to redundancy* in a calculus $\boldsymbol{\mathcal{S}}$, if every possible $\boldsymbol{\mathcal{S}}$-inference $\pi = C_1,.., C_k \vdash C$ from N is redundant w.r.t. N. A calculus $\boldsymbol{\mathcal{S}}$ is *refutationally complete with redundancy elimination* if for every clause set N that is saturated *up to redundancy* in $\boldsymbol{\mathcal{S}}$, we have $\square \in N$ if N is unsatisfiable.

It can be shown [see e.g., Kazakov, 2005] that the ordered resolution calculus $\mathcal{OR}^{\succ}_{\mathit{Sel}}$ is refutationally complete with redundancy elimination for every admissible ordering $\succ$ and every selection function *Sel*.

3.5.3 Equational Reasoning

Shortly after the resolution calculus has been introduced, Robinson & Wos [1969] have formulated an extension of this calculus with build-in equality. Equality plays a fundamental rôle in many applications of formal methods in mathematics and computer science, hence reasoning with equality has been and remains one of the central topics in automated deduction.

In this section we formulate two calculi with equality that are commonly used in automated theorem provers: the *ordered paramodulation calculus* [Robinson & Wos, 1969] and the *superposition calculus* [Bachmair & Ganzinger, 1990].

The Ordered Paramodulation Calculus

The *ordered paramodulation calculus* $\mathcal{OP}^{\succ}_{\mathit{Sel}}$ is an extension of the ordered resolution calculus $\mathcal{OR}^{\succ}_{\mathit{Sel}}$ with two rules given in System 4. Like ordered resolution, this calculus is parametrized by an ordering $\succ$ on literals and a selection function *Sel*, however now the ordering $\succ$ is defined on terms. In order to simplify the exposition

Ordered Paramodulation

$$\mathtt{OP}: \frac{C \vee \underline{\boldsymbol{s}} \simeq \boldsymbol{t}^{\star} \quad D \vee \boldsymbol{L}[\underline{\boldsymbol{s'}}]}{C\sigma \vee D\sigma \vee L[t]\sigma}$$

$\left[\textit{where (i) } \sigma = \mathrm{mgu}(s, s')\textit{; (ii) } s \simeq t \textit{ is eligible strictly maximal w.r.t. } C \textit{ and } \sigma\textit{; (iii) } L[s'] \textit{ is eligible (strictly maximal if positive) w.r.t. } D \textit{ and } \sigma\textit{; (iv) } (s\sigma \simeq t\sigma) \not\preceq L[s']\sigma\textit{; (v) } t\sigma \not\succeq s\sigma \textit{ and (vi) } s' \textit{ is not a variable.}\right]$

Reflexivity Resolution

$$\mathtt{RR}: \frac{C \vee \underline{\boldsymbol{s}} \not\simeq \underline{\boldsymbol{s'}}}{C\sigma}$$

$\left[\textit{where (i) } \sigma = \mathrm{mgu}(s, s') \textit{ and (ii) } s \not\simeq s' \textit{ is eligible w.r.t. } C \textit{ and } \sigma\textit{.}\right]$

System 4: The ordered paramodulation calculus $\mathcal{OP}^{\succ}_{\mathit{Sel}}$

of calculi with equality, we identify every *non-equational atom* A with equation $A \simeq \mathtt{T}$ (and its negation with $A \not\simeq \mathtt{T}$), where $\mathtt{T}$ is some fixed constant, which stands for “True”. This allows us to deal only with equational atoms of form $E_1 \simeq E_2$ over two sorts of expressions, where E_1, E_2 are either atoms or terms.

Definition 3.5.2. An ordering $\succ$ is *admissible for paramodulation* if $\succ$ is admissible for resolution (see Definition 3.5.1) and additionally:

(**T**) $\succ$ is total on ground terms with the least element $\mathtt{T}$;
(**E1**) $t \prec s \lhd L$ implies $L[s] \succ L[t]$ (monotonicity);
(**E2**) $t \prec s \lhd E_1$ implies $(E_1[s] \simeq E_2) \succ (s \simeq t)$; ◈

Condition (T) and (E1) of admissible ordering together with condition (L) from Definition 3.5.1 imply that $\succ$ is a total reduction ordering on ground expressions (see terminology on p. 57).

The Superposition Calculus

The *superposition calculus* $\mathcal{SP}^{\succ}_{Sel}$ extends the ordered resolution calculus $\mathcal{OR}^{\succ}_{Sel}$ with more refined inference rules for equality, given in System 5. Instead of a single Ordered Paramodulation rule we now have three inference rules: the Ordered Paramodulation rule into non-equational literals, the Positive Superposition rule for paramodulation into the maximal term of positive equations and the Negative Superposition rule for paramodulation into the maximal term of negative equations.[11] Additionally a special Equality Factoring rule is required to retain refutational completeness. Instead of this rule, one alternatively can use rule Merging Paramodulation formulated in Figure 3.2.

Figure 3.2 The merging paramodulation rule

Merging Paramodulation

$$\mathtt{MP} : \frac{C \vee \underline{\boldsymbol{s}} \boldsymbol{\simeq} \boldsymbol{t}^{\star} \quad D \vee \underline{u} \simeq v \vee \underline{\boldsymbol{r}} \boldsymbol{\simeq} \boldsymbol{h}[\underline{\boldsymbol{s}'}]^{\star}}{C\sigma \vee D\sigma \vee r\sigma \simeq h[t]\sigma \vee u\sigma \simeq v\sigma}$$

[*where (i)* $\sigma = mgu(\{s{=}s', r{=}u\})$*; (ii)* $s \simeq t$ *is eligible strictly maximal w.r.t.* C *and* σ*; (iii)* $r \simeq h$ *is eligible strictly maximal w.r.t.* $D \vee u \simeq v$ *and* σ*; (iv)* $t\sigma \not\succeq s\sigma$*; (v)* $h\sigma \not\succeq r\sigma$ *and (vi)* s' *is not a variable.*]

[11] Ordered Paramodulation into non-equational literals is often described as an instance of Positive Superposition or Negative Superposition when atoms are viewed as equations over two sorts of expressions

Ordered Paramodulation

$$\texttt{OP}: \frac{C \vee \underline{\boldsymbol{s}} \simeq \boldsymbol{t}^{\star} \quad D \vee \boldsymbol{L}[\underline{\boldsymbol{s}'}]}{C\sigma \vee D\sigma \vee L[t]\sigma}$$

[*where (i)* $\sigma = \mathrm{mgu}(s, s')$*; (ii)* $s \simeq t$ *is eligible strictly maximal w.r.t.* C *and* σ*; (iii)* $L[s']$ *is eligible (strictly maximal if positive) w.r.t.* D *and* σ*; (iv)* $L[s']$ *is a non-equational literal; (v)* $t\sigma \not\succeq s\sigma$ *and (vi)* s' *is not a variable.*]

Positive Superposition

$$\texttt{PS}: \frac{C \vee \underline{\boldsymbol{s}} \simeq \boldsymbol{t}^{\star} \quad D \vee \boldsymbol{r}[\underline{\boldsymbol{s}'}] \simeq \boldsymbol{h}^{\star}}{C\sigma \vee D\sigma \vee r[t]\sigma \simeq h\sigma}$$

[*where (i)* $\sigma = \mathrm{mgu}(s, s')$*; (ii)* $s \simeq t$ *is eligible strictly maximal w.r.t.* C *and* σ*; (iii)* $r \simeq h$ *is eligible strictly maximal w.r.t.* D *and* σ*; (iv)* $(s\sigma \simeq t\sigma) \not\succeq (r\sigma \simeq h\sigma)$*; (v)* $t\sigma \not\succeq s\sigma$*; (vi)* $h\sigma \not\succeq r\sigma$ *and (vii)* s' *is not a variable.*]

Negative Superposition

$$\texttt{NS}: \frac{C \vee \underline{\boldsymbol{s}} \simeq \boldsymbol{t}^{\star} \quad D \vee \boldsymbol{r}[\underline{\boldsymbol{s}'}] \not\simeq \boldsymbol{h}^{\star}}{C\sigma \vee D\sigma \vee r[t]\sigma \not\simeq h\sigma}$$

[*where (i)* $\sigma = \mathrm{mgu}(s, s')$*; (ii)* $s \simeq t$ *is eligible strictly maximal w.r.t.* C *and* σ*; (iii)* $r \not\simeq h$ *is eligible strictly maximal w.r.t.* D *and* σ*; (iv)* $t\sigma \not\succeq s\sigma$*; (v)* $h\sigma \not\succeq r\sigma$ *and (vi)* s' *is not a variable.*]

Reflexivity Resolution

$$\texttt{RR}: \frac{C \vee \underline{\boldsymbol{s}} \not\simeq \underline{\boldsymbol{s}'}}{C\sigma}$$

[*where (i)* $\sigma = \mathrm{mgu}(s, s')$ *and (ii)* $s \not\simeq s'$ *is eligible w.r.t.* C *and* σ*.*]

Equality Factoring

$$\texttt{EF}: \frac{C \vee \underline{s'} \simeq h \vee \underline{\boldsymbol{s}} \simeq \boldsymbol{t}^{\star}}{C\sigma \vee t\sigma \not\simeq h\sigma \vee s'\sigma \simeq h\sigma}$$

[*where (i)* $\sigma = \mathrm{mgu}(s, s')$*; (ii)* $s \simeq t$ *is eligible strictly maximal w.r.t.* $C \vee s' \simeq h$ *and* σ*, and (iii)* $t\sigma \not\succeq s\sigma$*.*]

System 5: The superposition calculus $\mathcal{SP}^{\succ}_{Sel}$

Definition 3.5.3. An ordering $\succ$ is *admissible for superposition* if $\succ$ is admissible for paramodulation (see Definition 3.5.2) and addittionally:

(**E3**) $s \succ t \succ h$ implies $(s \not\simeq h) \succ (s \simeq t)$, and

(**E4**) $s \succ t \succ h$ implies $(s \simeq t) \succ (t \not\simeq h)$. ◇

Both the *ordered paramodulation calculus* $\mathcal{OP}^{\succ}_{Sel}$ and the *superposition calculus* $\mathcal{SP}^{\succ}_{Sel}$ are refutationally complete with redundancy elimination for every choice of a selection function *Sel* and an admissible ordering $\succ$.

3.5.4 Chaining Calculi

The ordered paramodulation and superposition calculi described in the previous section, are examples of calculi with *build-in theory*, in these cases, the theory of equality. It is in principle possible to use the resolution calculus for reasoning with equality, by processing explicitly all necessary equational axioms. However, this approach is impractical, as it gives a huge number of unnecessary inferences. A similar situation may happen when other useful theories are considered.

Bachmair & Ganzinger [1995, 1998*b*] have studied automated reasoning over theories induced by a collection of so-called *compositional axioms* of form:

$$xSy \wedge yTz \rightarrow xHz \quad \text{or short} \quad S \circ T \subseteq H \tag{3.19}$$

The simplest instance of this scheme is the *transitivity axiom*:

$$xTy \wedge yTz \rightarrow xTz \quad \text{or short} \quad T \circ T \subseteq T \tag{3.20}$$

We have already seen in subsection 2.5.2 that resolution inferences with transitivity axioms are hard to control. Yet many of such inferences can be shown to be *redundant*: see [Nieuwenhuis & Rubio, 2001, Example 4.10] and [Kazakov, 2005, Example 3.38]. Bachmair & Ganzinger [1995, 1998*b*] have formulated several *chaining* calculi, where only necessary inferences with compositional axioms (3.19) are performed using dedicated inference rules. Below we describe the most general of these calculi and formulate sufficient conditions for their admissible parameters.

Reasoning with Compositional Binary Relations

Let $\boldsymbol{\mathcal{C}}$ be a *compositional theory* induced by a collection of compositional axioms (denoted by the same letter) of form (3.19). The predicate symbols involved in $\boldsymbol{\mathcal{C}}$ are called *special predicate symbols*, and, like for equality, we will use the infix notation to distinguish them from other predicate symbols.

A compositional theory $\boldsymbol{\mathcal{C}}$ is *associative* if for every $(S \circ T \subseteq U) \in \boldsymbol{\mathcal{C}}$ and $(U \circ H \subseteq W) \in \boldsymbol{\mathcal{C}}$ there exists a special predicate symbol V such that $(T \circ H \subseteq V) \in \boldsymbol{\mathcal{C}}$ and $(S \circ V \subseteq W) \in \boldsymbol{\mathcal{C}}$, or in symbols: $(S \circ T) \circ H \ = \ S \circ (T \circ H)$. Most reasonable compositional theories are associative, including the theory of transitivity. The following *theory of a quasi-ordering* considered in Bachmair & Ganzinger [1995, 1998*b*], is another example of an associative compositional theory:

$$\begin{array}{lll} \succ \circ \succ \subseteq \succ; & \succsim \circ \succsim \subseteq \succsim; & \sim \circ \sim \subseteq \sim; \\ \succ \circ \succsim \subseteq \succ; & \succ \circ \sim \subseteq \succ; & \succsim \circ \sim \subseteq \succsim; \\ \succsim \circ \succ \subseteq \succ; & \sim \circ \succ \subseteq \succ; & \sim \circ \succsim \subseteq \succsim. \end{array} \tag{3.21}$$

The ordered chaining calculus $\mathcal{OC}^{\succsim}_{Sel}$ for an associative compositional theory $\boldsymbol{\mathcal{C}}$ is formulated in System 6. In all inference rules we assume that S, T and H are special predicate symbols such that $(S \circ T \subseteq H) \in \boldsymbol{\mathcal{C}}$. One can notice similarities between the ordered chaining calculus $\mathcal{OC}^{\succsim}_{Sel}$ and the superposition calculus $\mathcal{SP}^{\succsim}_{Sel}$ defined in System 5. In particular, analogies between the Ordered Chaining and the Positive

Ordered Chaining

$$\mathtt{OC}: \frac{C \vee \boldsymbol{tS\underline{s}}^{\star} \quad D \vee \boldsymbol{\underline{s}'Tt'}^{\star}}{C\sigma \vee D\sigma \vee t\sigma H t'\sigma}$$

[*where* (i) $\sigma = \mathrm{mgu}(s, s')$; (ii) tSs *is eligible strictly maximal w.r.t.* C *and* σ; (iii) $s'Tt'$ *is eligible strictly maximal w.r.t.* D *and* σ; (iv) $t\sigma \not\succ s\sigma$; (iv)$'$ $t\sigma \neq s\sigma$ *if* $H = T$; (v) $t'\sigma \not\succ s'\sigma$ *and* (v)$'$ $t'\sigma \neq s'\sigma$ *if* $H = S$.]

Negative Chaining

$$\mathtt{NC}: \frac{C \vee \boldsymbol{\underline{s}St}^{\star} \quad \neg(\boldsymbol{\underline{s}'Hh}) \vee D}{C\sigma \vee \neg(t\sigma T h\sigma) \vee D\sigma} \qquad \frac{C \vee \boldsymbol{tT\underline{s}}^{\star} \quad \neg(\boldsymbol{hH\underline{s}'}) \vee D}{C\sigma \vee \neg(h\sigma S t\sigma) \vee D\sigma}$$

[*where* (i) $\sigma = \mathrm{mgu}(s, s')$; (ii) sSt *is eligible strictly maximal w.r.t.* C *and* σ; (iii) $\neg(s'Hh)$ *is eligible w.r.t.* D *and* σ; (iv) $t\sigma \not\succeq s\sigma$ *and* (v) $h\sigma \not\succeq s'\sigma$.]

[*where* (i) $\sigma = \mathrm{mgu}(s, s')$; (ii) tTs *is eligible strictly maximal w.r.t.* C *and* σ; (iii) $\neg(hHs')$ *is eligible w.r.t.* D *and* σ; (iv) $t\sigma \not\succeq s\sigma$ *and* (v) $h\sigma \not\succ s'\sigma$.]

Compositional Resolution

$$\mathtt{CR}: \frac{C \vee \underline{s}Hh \vee \boldsymbol{\underline{s}''S't'}^{\star} \quad D \vee \boldsymbol{\underline{s}'St}^{\star}}{D\sigma \vee \neg(t\sigma T h\sigma) \vee s\sigma H h\sigma} \qquad \frac{C \vee hH\underline{s} \vee \boldsymbol{t'S'\underline{s}''}^{\star} \quad D \vee \boldsymbol{tT\underline{s}'}^{\star}}{D\sigma \vee \neg(h\sigma S t\sigma) \vee h\sigma H s\sigma}$$

[*where* (i) $\sigma = \mathrm{mgu}(\{s{=}s', s{=}s''\})$; (ii) $s''S't'$ *is eligible strictly maximal w.r.t.* $C \vee sHh$ *and* σ; (iii) $s'St$ *is eligible strictly maximal w.r.t.* D *and* σ; (iv) $s''\sigma S't'\sigma \not\succ s'\sigma St\sigma$; (v) $t\sigma \not\succeq s'\sigma$ *and* (vi) $h\sigma \not\succeq s\sigma$.]

[*where* (i) $\sigma = \mathrm{mgu}(\{s{=}s', s{=}s''\})$; (ii) $t'S's''$ *is eligible strictly maximal w.r.t.* $C \vee hHs$ *and* σ; (iii) tTs' *is eligible strictly maximal w.r.t.* D *and* σ; (iv) $s''\sigma S't'\sigma \not\succ t\sigma Ts'\sigma$; (v) $t\sigma \not\succeq s'\sigma$ *and* (vi) $h\sigma \not\succeq s\sigma$.]

System 6: The ordered chaining calculus for compositional binary relations $\mathcal{OC}^{\succ}_{Sel}$

Superposition rule, and between Negative Chaining and Negative Superposition rules. The Compositional Resolution rules have a similar function as Reflexivity Resolution.

Definition 3.5.4. An ordering $\succ$ on expressions is called *admissible for chaining* if $\succ$ is *admissible for resolution* (see Definition 3.5.1) and additional:

(**T**) $\succ$ is total on ground terms,

and for every axiom $(S \circ T \subseteq H) \in \boldsymbol{\mathcal{C}}$ and terms $s \succ t$, $s \succ h$ we have:

(**C1**) $\neg(sHh) \succ (sSt)$; $\quad \neg(hHs) \succ (tTs)$; $\quad \neg(sHs) \succ (tTs)$;
(**C2**) $\neg(sHh) \succ \neg(tTh)$; $\neg(hHs) \succ \neg(hSt)$; $\neg(sHs) \succ \neg(sSt)$;
(**C3**) $\quad (sSt) \succ \neg(tTh)$; $\quad (tTs) \succ \neg(hSt)$; $\quad (sHs) \succ (sSt)$ ⟐

Remark 3.5.5. In their original chaining calculi, Bachmair & Ganzinger [1998*b*] did not allow for arbitrary associative compositional axioms, but for those induced by a total precedence $\gg$ on special predicate symbols. Given a precedence $\gg$ on special predicate symbols, we say that a compositional theory $\boldsymbol{\mathcal{C}}$ is *induced by* $\gg$ if

$\mathcal{C} = \{S \circ T \subseteq H \mid H = max_{\gg}(S,T)\}$. Although the theory of transitivity (3.20) and the theory of a quasi-ordering (3.21) fulfil this property, there are some natural associative theories which lack it. In particular, the *theory of metric distances* that will be considered in subsection 5.1.1. Hence, our formulation of chaining calculus is a non-trivial generalisation over those given in [Bachmair & Ganzinger, 1998*b*]. ◈

The Subterm Chaining Calculus

It is possible to describe a hybrid calculus which incorporates both a theory of compositional axioms $\mathcal{C}$ and the theory of equality. One could simply use the Ordered Paramodulation rule to perform paramodulation inferences into non-equational literals (including all special non-equational literals) as before. However it is possible to use the advantage of compositional theory and perform paramodulation inferences only *into the largest argument* of special literals. Essentially, equality is treated as a part of a compositional theory: we assume that every compositional theory $\mathcal{C}$ contains all axioms $\simeq \circ\, S \subseteq S$ and $S \circ \simeq\, \subseteq S$ for every special predicate symbol S. *However we do not allow the equational predicate to be the result of composition of non-equational predicates, i.e.,* $S \circ T \subseteq\, \simeq$ *implies that* $S = \simeq$ *and* $T = \simeq$. Applying these modifications, we obtain a so-called *subterm chaining calculus* that is an extension of the ordered chaining calculus from System 6 with inference rules given in System 7, where now equality can be used as a compositional predicate symbol. Note that the Positive Superposition and the Negative Superposition rules are instances of the Ordered Subterm Chaining and the Negative Subterm Chaining rules respectively, when $S = \simeq$ (in the left variants). The right variants of these rules are not needed for equational literals, since we treat equality symmetrically. The Equality Factoring rule is simulated by a self-application of the Compositional Resolution rule for equational atoms.

Definition 3.5.6. The ordering $\succ$ on expressions is called *admissible for subterm chaining* if $\succ$ is *admissible for chaining* (see Definition 3.5.6) and superposition (see Definition 3.5.3). ◈

Redundancy

The standard redundancy criterion formulated in subsection 3.5.2 that uses an admissible ordering $\succ$, does not always work for the chaining calculi. We will not discuss the reasons for this here, and forward the reader to [Kazakov, 2005] for details. For proving redundancy we have to use an ordering $\succ$ which is slightly weaker than the admissible ordering $\succ$ of the calculus (so less clauses and inferences might be redundant):

Ordered Paramodulation

$$\texttt{OP}: \frac{C \vee \underline{\boldsymbol{s}} \boldsymbol{\simeq} \boldsymbol{t}^\star \quad D \vee \boldsymbol{L}[\underline{\boldsymbol{s'}}]}{C\sigma \vee D\sigma \vee L[t]\sigma}$$

[*where (i)* $\sigma = \mathrm{mgu}(s, s')$; *(ii)* $s \simeq t$ *is eligible strictly maximal w.r.t.* C *and* σ; *(iii)* $L[s']$ *is eligible (strictly maximal if positive) w.r.t.* D *and* σ; *(iv)* $L[s']$ *is a* *non-special literal;* *(v)* $t\sigma \not\succeq s\sigma$ *and (vi)* s' *is not a variable.*]

Reflexivity Resolution

$$\texttt{RR}: \frac{C \vee \underline{\boldsymbol{s}} \boldsymbol{\not\simeq} \underline{\boldsymbol{s'}}}{C\sigma}$$

[*where (i)* $\sigma = \mathrm{mgu}(s, s')$ *and (ii)* $s \not\simeq s'$ *is eligible w.r.t.* C *and* σ.]

Ordered Subterm Chaining

$$\texttt{OSC}: \frac{C \vee \underline{\boldsymbol{s}} \boldsymbol{\simeq} \boldsymbol{t}^\star \quad \boldsymbol{r}[\underline{\boldsymbol{s'}}]\boldsymbol{S}\boldsymbol{h}^\star \vee D}{C\sigma \vee r[t]\sigma S h\sigma \vee D\sigma}$$

[*where (i)* $\sigma = \mathrm{mgu}(s, s')$; *(ii)* $s \simeq t$ *is eligible strictly maximal w.r.t.* C *and* σ; *(iii)* rSh *is eligible strictly maximal w.r.t.* D *and* σ; *(iv)* $(s\sigma \simeq t\sigma) \not\succeq (r\sigma S h\sigma)$; *(v)* $t\sigma \not\succeq s\sigma$; *(vi)* $h\sigma \not\succeq r\sigma$ *and (vii)* s' *is not a variable.*]

$$\frac{C \vee \underline{\boldsymbol{s}} \boldsymbol{\simeq} \boldsymbol{t}^\star \quad \boldsymbol{h}\boldsymbol{S}\boldsymbol{r}[\underline{\boldsymbol{s'}}]^\star \vee D}{C\sigma \vee h\sigma S r[t]\sigma \vee D\sigma}$$

[*where (i)* $\sigma = \mathrm{mgu}(s, s')$; *(ii)* $s \simeq t$ *is eligible strictly maximal w.r.t.* C *and* σ; *(iii)* hSr *is eligible strictly maximal w.r.t.* D *and* σ; *(iv)* $(s\sigma \simeq t\sigma) \not\succeq (h\sigma S r\sigma)$; *(v)* $t\sigma \not\succeq s\sigma$; *(vi)* $h\sigma \not\succ r\sigma$; *(vi)'* $S \neq \simeq$, *and (vii)* s' *is not a variable.*]

Negative Subterm Chaining

$$\texttt{NSC}: \frac{C \vee \underline{\boldsymbol{s}} \boldsymbol{\simeq} \boldsymbol{t}^\star \quad \boldsymbol{\neg(r}[\underline{\boldsymbol{s'}}]\boldsymbol{Sh)} \vee D}{C\sigma \vee \neg(r[t]\sigma S h\sigma) \vee D\sigma}$$

[*where (i)* $\sigma = \mathrm{mgu}(s, s')$; *(ii)* $s \simeq t$ *is eligible strictly maximal w.r.t.* C *and* σ; *(iii)* $\neg(rSh)$ *is eligible w.r.t.* D *and* σ; *(iv)* $t\sigma \not\succeq s\sigma$; *(v)* $h\sigma \not\succeq r\sigma$ *and (vi)* s' *is not a variable.*]

$$\frac{C \vee \underline{\boldsymbol{s}} \boldsymbol{\simeq} \boldsymbol{t}^\star \quad \boldsymbol{\neg(hSr}[\underline{\boldsymbol{s'}}]\boldsymbol{)} \vee D}{C\sigma \vee \neg(h\sigma S r[t]\sigma) \vee D\sigma}$$

[*where (i)* $\sigma = \mathrm{mgu}(s, s')$; *(ii)* $s \simeq t$ *is eligible strictly maximal w.r.t.* C *and* σ; *(iii)* $\neg(hSr)$ *is eligible w.r.t.* D *and* σ; *(iv)* $t\sigma \not\succeq s\sigma$; *(v)* $h\sigma \not\succ r\sigma$; *(v)'* $S \neq \simeq$, *and (vi)* s' *is not a variable.*]

System 7: The subterm chaining calculus $\mathcal{SC}^{\succ}_{Sel}$

Definition 3.5.7 (Redundancy Ordering)**.** A *redundancy ordering* for an admissible ordering $\succ$ (for chaining calculi) is a partial ordering $\mathrel{\succ\!\!\!-}$ on ground literals such that the following conditions hold:

(**CRO1**) $\mathrel{\succ\!\!\!-} \subseteq \succ$, $\quad \succ \circ \mathrel{\succ\!\!\!-} \subseteq \succ$, $\quad \mathrel{\succ\!\!\!-} \circ \succ \subseteq \succ$;
(**CRO2**) $S \circ T \subseteq H$, $L \mathrel{\succ\!\!\!-} tSs$, $L \mathrel{\succ\!\!\!-} sTh$ and $s \succeq t$, $s \succeq h$
imply that $\quad L \mathrel{\succ\!\!\!-} tHh$ ⟐

Intuitively, the redundancy ordering $\mathrel{\succ\!\!\!-}$ is a subordering of $\succ$ under which the rule Ordered Chaining is *monotone* (in some weak sense) w.r.t. $\mathrel{\succ\!\!\!-}$. It is possible to show that the chaining calculi are refutationally complete with this notion of redundancy for every selection function *Sel*, every admissible ordering $\succ$, and every redundancy ordering $\mathrel{\succ\!\!\!-}$ for $\succ$ [see Kazakov, 2005].

3.5.5 Variations of Inference Systems

Many variations of inference rules in saturation-based calculi have been proposed in the literature, that intend to restrict or speed-up inferences for certain cases. In this section, we describe two particular classes of such inference rules, namely *simulteneous paramodulation* and *hyper-inferences*.

Simultaneous Paramodulation and Superposition

The inference rules of the ordered paramodulation and superposition calculi can be modified in such a way that the replacement is performed in several equal subterms *simultaneously*: see Figure 3.3. These variants are typically employed in theorem provers instead of the original rules, because this would make it more efficient for shared data-structures. In section 4.3 we demonstrate how these rules allow one to obtain decision procedures with optimal complexities.

Figure 3.3 The simultaneous paramodulation and superposition rules

(Simulteneous) Ordered Paramodulation

$$\texttt{OP} : \frac{C \vee \underline{\boldsymbol{s}} \simeq \boldsymbol{t}^{\star} \quad D[\underline{s'}] \vee \boldsymbol{L}[\underline{\boldsymbol{s'}}]}{C\sigma \vee D[t]\sigma \vee L[t]\sigma}$$

[*where (i)* $\sigma = \mathrm{mgu}(s, s')$; *(ii)* $s \simeq t$ *is eligible strictly maximal w.r.t.* C *and* σ; *(iii)* $L[s']$ *is eligible (strictly maximal if positive) w.r.t.* D *and* σ; *(iv)* $(s\sigma \simeq t\sigma) \not\succeq L[s']\sigma$; *(v)* $t\sigma \not\succeq s\sigma$ *and (vi)* s' *is not a variable.*]

(Simulteneous) Positive Superposition

$$\texttt{PS} : \frac{C \vee \underline{\boldsymbol{s}} \simeq \boldsymbol{t}^{\star} \quad D[\underline{s'}] \vee \boldsymbol{r}[\underline{\boldsymbol{s'}}] \simeq \boldsymbol{h}[\underline{\boldsymbol{s'}}]^{\star}}{C\sigma \vee D[t]\sigma \vee r[t]\sigma \simeq h[t]\sigma}$$

[*where (i)* $\sigma = \mathrm{mgu}(s, s')$; *(ii)* $s \simeq t$ *is eligible strictly maximal w.r.t.* C *and* σ; *(iii)* $r \simeq h$ *is eligible strictly maximal w.r.t.* D *and* σ; *(iv)* $(s\sigma \simeq t\sigma) \not\succeq (r\sigma \simeq h\sigma)$; *(v)* $t\sigma \not\succeq s\sigma$; *(vi)* $h\sigma \not\succeq r\sigma$ *and (vii)* s' *is not a variable.*]

(Simulteneous) Negative Superposition

$$\texttt{NS} : \frac{C \vee \underline{\boldsymbol{s}} \simeq \boldsymbol{t}^{\star} \quad D[\underline{s'}] \vee \boldsymbol{r}[\underline{\boldsymbol{s'}}] \not\simeq \boldsymbol{h}[\underline{\boldsymbol{s'}}]^{\star}}{C\sigma \vee D[t]\sigma \vee r[t]\sigma \not\simeq h[t]\sigma}$$

[*where (i)* $\sigma = \mathrm{mgu}(s, s')$; *(ii)* $s \simeq t$ *is eligible strictly maximal w.r.t.* C *and* σ; *(iii)* $r \not\simeq h$ *is eligible strictly maximal w.r.t.* D *and* σ; *(iv)* $t\sigma \not\succeq s\sigma$; *(v)* $h\sigma \not\succeq r\sigma$ *and (vi)* s' *is not a variable.*]

Hyper-Inferences

A *hyper-inference rule* is a rule composed from several ordinary inference rules. A typical example of such rule is the well-known Ordered Hyper-resolution rule. In Figure 3.4 we formulated a variant of this rule with selection, which can be seen as simulteneous application of n resolution inferences.

We say that a multiset $D' = \{\neg B_1,..,\neg B_n\}_m$ of negative literals is *eligible w.r.t. D and σ (for hyper-resolution)* if either (***a***) $D' = \mathit{Sel}(D \vee D') \neq \{\}_m$, or, otherwise (***b***) $\mathit{Sel}(D \vee D') = \{\}_m$, $n = 1$ and $\neg B_1\sigma$ is maximal w.r.t. $D\sigma$. Now the *Ordered Hyper-Resolution calculus with selection $\mathcal{HR}^{\succ}_{\mathit{Sel}}$* is defined by replacing the Ordered Resolution rule with the Ordered Hyper-resolution rule.

Figure 3.4 The hyper-resolution rule

Ordered Hyper-resolution

$$\mathtt{HR}: \frac{C_1 \vee \underline{\boldsymbol{A_1}}^{\star} \quad \ldots \quad C_n \vee \underline{\boldsymbol{A_n}}^{\star} \qquad \neg\underline{\boldsymbol{B_1}} \vee \cdots \vee \neg\underline{\boldsymbol{B_n}} \vee D}{C_1\sigma \vee \cdots \vee D\sigma}$$

[*where (i) $\sigma = mgu(\{A_1{=}B_1, \ldots, A_n{=}B_n\})$; (ii) $A_i\sigma$ are eligible strictly maximal w.r.t. C_i and σ, $i = 1,..,n$ and (iii) $\{\neg B_1,..,\neg B_n\}_m$ is eligible w.r.t. D and σ.*]

Hyper-inferences are typically used to avoid generation of unnecessary intermediate clauses in inferences. However such rules are also more restrictive, since all conditions of a hyper-inference must be satisfied *simultaneously.* In Figure 3.5, we extend this principle and formulate a Negative Hyper-Chaining rule. This rule is applied to a clause whose all selected literals are special and for each of them a Negative Chaining inference with the same clause is possible. Then all these inferences are applied at once provided that all ordering restriction can be satisfied *simultaneously.* Here by $x\tilde{S}y$ we denote a special atom xSy or its inverse ySx, to avoid formulation of the symmetric version of the rule. Given a clause with several selected literals, we allow either (***i***) a simultaneous application of the negative chaining rule on *all* of these literals, or (***ii***) any other inference with *any* of the selected literals.

Figure 3.5 A hyper- extension of the Negative Chaining rule

Negative Hyper-Chaining

$$\mathtt{HC}: \frac{C_i \vee \underline{\boldsymbol{s_1}}\boldsymbol{\tilde{S}_1 t_1}^{\star} \ \ldots\ C_j \vee \underline{\boldsymbol{s_n}}\boldsymbol{\tilde{S}_n t_n}^{\star} \quad \neg(\underline{\boldsymbol{s'_1}}\boldsymbol{\tilde{H}_1 h_1})^{\sharp} \vee \cdots \vee \neg(\underline{\boldsymbol{s'_n}}\boldsymbol{\tilde{H}_n h_n})^{\sharp} \vee D}{C_i\sigma \vee \neg(t_1\sigma\tilde{T}_1 h_1\sigma) \vee D\sigma}$$

[*where (i) $\tilde{S}_i \circ \tilde{T}_i \subseteq \tilde{H}_i$ for all $1 \leq i \leq n$; (ii) $\sigma = mgu(\{s_i{=}s'_i \mid 1 \leq i \leq n\})$; and there is a ground substitution $\sigma^0 = \sigma\tau^0$ such for all i with $1 \leq i \leq n$: (iii) $s_i S_i t_i$ is eligible strictly maximal w.r.t. C_i and σ^0; (iv) $\{\neg(s'_1 H_1 h_1),..,\neg(s'_n \tilde{H}_n h_n)\}_m$ is eligible w.r.t. D and σ^0; (v) $t_i\sigma^0 \not\preceq s_i\sigma^0$ and (vi) $h_i\sigma^0 \not\preceq s'_i\sigma^0$.*]

3.5.6 The Theorem Proving Process

So far we have described saturation-based calculi from the static point of view, i.e., as a collection of inference rules. We have also formulated some redundancy criteria, but did not discuss how they can be used. In this section we are concerned with the question of how deduction of clauses and redundancy elimination is organised in theorem provers.

Simplification Rules

Deletion of redundant clauses in modern theorem provers is organised using special *simplification rules*. In most cases simplification rules do not just delete redundant clauses, but rather replace clauses with "simpler" ones, which is why they are called so. A general *simplification rule* typically has the following form:

A Simplification Rule

$$\mathtt{SR}: \frac{S \cup [\![S']\!]}{S_1 \parallel \cdots \parallel S_n} \tag{3.22}$$

$$\Big[\textit{where the conditions of the rule hold}\Big]$$

where S, S', $S_1, .., S_n$ are clause sets, $[\![S']\!]$ are the clauses that should be deleted after the inference is applied, and $S_1, \ldots, S_n$ are the possible (non-deterministic) choices for the clauses produced by the inference. In Figure 3.6 we have listed some simplification rules commonly used in saturation-based theorem provers.

Figure 3.6 Some simplification rules used in saturation-based theorem provers

Tautology Deletion

$$\mathtt{TD}: \frac{[\![C \vee A \vee \neg A]\!]}{}$$

$$\Big[\quad - \quad\Big]$$

Elimination of Duplicate Literals

$$\mathtt{ED}: \frac{[\![C \vee L \vee L]\!]}{C \vee L}$$

$$\Big[\quad - \quad\Big]$$

Subsumption Deletion

$$\mathtt{SD}: \frac{C \quad [\![D]\!]}{}$$

$$\Big[\textit{where (i) } C \textit{ strictly subsumes } D\Big]$$

Splitting

$$\mathtt{SP}: \frac{[\![C \vee D]\!]}{C \parallel D}$$

$$\Big[\textit{where (i) } C \neq \Box\textit{; (ii) } D \neq \Box \textit{ and (iii) } \mathrm{vars}[C] \cap \mathrm{vars}[D] = \{\}.\Big]$$

Simplification rules should be *admissible*, which amounts to the following two conditions: (**1**) the rule should be *sound*, i.e., there must be no way of producing an unsatisfiable clause set from a satisfiable one, and (**2**) deletion of clauses should be done according to the *redundancy criterion*, i.e., every deleted clause should follow logically from some smaller clauses of the rule. Admissible simplification rules can be freely used in any calculus without affecting its refutaional completenes. Therefore, we often call such rules as *optional* ones.

In our decision procedures we will introduce custom simplification rules to achieve certain desirable effects. In subsection 2.5.2 we have used one of such rules for treating clauses for compositional axioms, namely:

Splitting through New Predicate Symbol

$$\texttt{SPP}:\ \frac{C \vee D}{\begin{array}{l} C \vee \ \ u_C(\overline{x}) \\ D \vee \neg u_C(\overline{x}) \end{array}} \tag{3.23}$$

$$\left[\begin{array}{l}\textit{where (i) } C \neq \square\textit{; (ii) } D \neq \square\textit{; (iii) } \mathsf{vars}[C] \cap \mathsf{vars}[D] = \overline{x} \textit{ and} \\ \textit{(iv) } u_C \textit{ is an extended predicate symbol introduced for } C.\end{array}\right]$$

Unlike those rules given in Figure 3.6, this rule extends the original signature with a *fresh* predicate symbol u_C introduced for clause C. For proving soundness of such a rule, one usually needs to describe how the model for the conclusion of this rule can be obtained from the model for its premises. For this particular rule one needs to expand the model by interpreting $u_C(\overline{x})$ as a first-order formula $\exists\overline{y}(\neg C)$, where $\overline{y} = \mathsf{vars}[C] \setminus \overline{x}$. A similar construction can be used for proving soundness of other simplification rules extending the signature, which we introduce later.

A Model of a Saturation Process

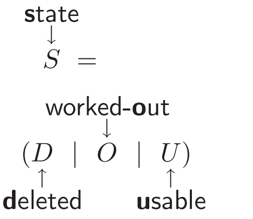
In order to estimate the complexity of different saturation-based procedures, we will use a simple model of computation for a refutational-based theorem prover. Similar, but more detailed and more practically-oriented models can be found, for instance, in [Weidenbach, 2001; Bachmair & Ganzinger, 2001].

We assume that we are given a collection $\boldsymbol{\mathcal{S}}$ of *inference rules* of a calculus augmented with some (possibly non-deterministic) *simplification rules*. Among simplification rules we distinguish a subset $\boldsymbol{\mathcal{S}}_e$ of *eager simplification rules*. A distinction between these two types of simplification rules will be clarified in a moment.

A *state* of our saturation-based theorem prover is a set of clauses S partitioned into three pairwise disjoint subsets D, O and U of *deleted clauses*, *worked-out clauses* and *usable clauses* respectively. This is written shortly as $S = (D \mid O \mid U)$. Given an input set of clauses N to be processed, a state of the prover is initialised to

$S := (\{\} \mid \{\} \mid N)$. After that, the current set of clauses is a subject to *normalisation* and *deduction* steps according to the procedure in Figure 3.7.

Figure 3.7 A model of a prover with eager simplification rules



```
Prover ($N$)
    $S$:=Normalise ($\{\} \mid \{\} \mid N$);
    while ($U \neq \{\}$ and $\Box \notin U$) do
        $S$:=Normalise (Deduce ($S$));
    end;
    if ($U = \{\}$) then return ("Satisfiable");
    if ($\Box \in U$) then return ("Unsatisfiable");
end.
```

```
Normalise ($S$)
    repeat
        $S'$:= $S$;
        $S$:= Simplify ($S'$);
    until ($S = S'$);
    return($S$);
end.
```

Deduce

$$D \mid O \sqcup O' \mid \{C\} \sqcup U \;\Rightarrow\; (D \cup O' \mid O \cup \{C\} \mid U) \cup N \quad \text{where} \quad O \cup \{C\} \cup [\![O']\!] \vdash_{\mathcal{S}} N$$

$$D \mid O \sqcup O' \mid \{C\} \sqcup U \;\Rightarrow\; (D \cup O' \cup \{C\} \mid O \mid U) \cup N \quad \text{where} \quad O \cup [\![\{C\} \cup O']\!] \vdash_{\mathcal{S}} N$$

Simplify

$$D \mid O \sqcup O' \mid U \sqcup U' \;\Rightarrow\; (D \cup O' \mid O \mid U) \cup N \quad \text{where} \quad O \cup U \cup [\![O' \cup U']\!] \vdash_{\mathcal{S}_e} N$$

In the *deduction step* (function Deduce(·)), the next theorem proving state is obtained from the previous one by (**1**) selecting a usable clause, (**2**) inserting conclusions of all inferences between this clause and worked-out clauses, (**3**) moving deleted clauses into D and (**4**) moving the selected clause into the set of worked-out clauses O if it has not been deleted – see rule Deduce. In the *normalisation step* (function Normalise(·)), an *exhaustive* application of *eager* simplification rules according to rule Simplify is performed.

In the following, we assume that (***i***) every inference from $\boldsymbol{\mathcal{S}}$ can be computed in polynomial time in the size of its premises and (***ii***) normalisation of a clause set N w.r.t. eager simplification rules can be done in polynomial time in the size of N. These assumptions are reasonable: every rule from the calculi introduced so far satisfies property (*i*), and normalisation using all rules from Figure 3.6, but Subsumption Deletion, enjoys property (*ii*)[12].

Now we give a formula which will be used for calculation of the running times for saturation procedures that we describe. The time complexity is estimated in terms of the following parameters:

[12] For many clause classes, including those considered in this thesis, deletion of subsumed clauses can be also done in polynomial time

$|N|$ - *the size of the initial clause set;*
$\boldsymbol{c}$ - *the maximal number of normalised clauses;*
$\boldsymbol{s}$ - *the maximal number of clauses in a normalised clause set;*
$\boldsymbol{m}$ - *the maximal size of a normalised clause;*
$\boldsymbol{k}$ - *the maximal number of premises in all deduction and simplification rules.*

Given the values for these parameters, the total (possibly non-deterministic) running time for a saturation procedure is bounded by:

$$\boxed{\boldsymbol{t} \ = \ p(|N|) + \boldsymbol{c} \cdot p(\boldsymbol{m} \cdot \boldsymbol{s}^{(\boldsymbol{k}-1)})} \tag{3.24}$$

where $p(\cdot)$ is a polynomial function. Please see [Kazakov, 2005] for the details of how this formula has been derived.

3.5.7 Clause Normal Form Transformation

As has been said in the very beginning, saturation-based theorem provers do not operate with arbitrary first-order formulas, but with clauses. To use a theorem prover for first-order formulas, it is required, in the first place, to transform them into a *clause normal form* (short **CNF**). In this section we describe the standard **CNF**-transformation procedures, and discuss some complexity issues.

CNF-transformation has been thoroughly studied in literature [for the overview see Nonnengart & Weidenbach, 2001; Baaz, Egly & Leitsch, 2001]. The traditional way of producing **CNF**'s for decidable first-order fragments consists of three main steps. First, a formula is translated into a *negation normal form* by pushing negation inwards as far as possible. Second, a so-called *structural transformation* is applied to a formula, that splits the formula into a conjunction of *simple* formulas. In the last step, *skolemization* is employed that introduces Skolem functions for existentially quantified variables. After this step the universal quantifiers are dropped and the result is written in a clause form.

Negation Normal Form Transformation

A first-order formula (involving conjunction, disjunction and negation as the only Boolean connectives) is in *negation normal form* (or shortly **NNF**) if negation symbol appears only in front of atoms in this formula. For putting a first-oder formula into **NNF**, one has to distribute negation over Boolean connectives and quantifiers using the usual *de-Morgan*'s laws:

$$\begin{array}{ccc} \neg(A \wedge B) \Rightarrow \neg A \vee \neg B & & \neg\forall x.A \Rightarrow \exists x.\neg A \\ & \neg\neg A \ \Rightarrow \ A & \\ \neg(A \vee B) \Rightarrow \neg A \wedge \neg B & & \neg\exists x.A \Rightarrow \forall x.\neg A \end{array} \tag{3.25}$$

Formally the result $[F]^{nnf}$ of **NNF**-transformation for a formula F can be defined recursively over the definition of first-order formulas as in Figure 3.8. Note that the set of first-order formulas in negation normal form can be defined by the grammar:

Figure 3.8 Negation normal form transformation for first-order formulas

$$\mathcal{FO} ::= A \mid F_1 \vee F_2 \mid F_1 \wedge F_2 \mid \neg F_1 \mid \forall y.F_1 \mid \exists y.F_1.$$

$$\begin{array}{rcll@{\qquad}rcll}
[F]^{nnf} := \quad [A]^{nnf} & = & A & \mid & [F]^{nnf}_{\neg} := [A]^{nnf} & = & \neg A & \mid \\
[F_1 \wedge F_2]^{nnf} & = & [F_1]^{nnf} \wedge [F_2]^{nnf} & \mid & [F_1 \wedge F_2]^{nnf}_{\neg} & = & [F_1]^{nnf}_{\neg} \vee [F_2]^{nnf}_{\neg} & \mid \\
[F_1 \vee F_2]^{nnf} & = & [F_1]^{nnf} \vee [F_2]^{nnf} & \mid & [F_1 \vee F_2]^{nnf}_{\neg} & = & [F_1]^{nnf}_{\neg} \wedge [F_2]^{nnf}_{\neg} & \mid \\
[\neg F_1]^{nnf} & = & [F_1]^{nnf}_{\neg} & \mid & [\neg F_1]^{nnf}_{\neg} & = & [F_1]^{nnf} & \mid \\
[\forall y.F_1]^{nnf} & = & \forall y.[F_1]^{nnf} & \mid & [\forall y.F_1]^{nnf}_{\neg} & = & \exists y.[F_1]^{nnf}_{\neg} & \mid \\
[\exists y.F_1]^{nnf} & = & \exists y.[F_1]^{nnf}\ . & & [\exists y.F_1]^{nnf}_{\neg} & = & \forall y.[F_1]^{nnf}_{\neg}\ . &
\end{array}$$

$$[\mathcal{FO}]^{nnf} ::= A \mid \neg A \mid F_1 \vee F_2 \mid F_1 \wedge F_2 \mid \forall y.F_1 \mid \exists y.F_1. \tag{3.26}$$

where A is an atom and $F_1,\ F_2 \in [\mathcal{FO}]^{nnf}$.

Proposition 3.5.8. *For every formula $F \in \mathcal{FO}$ the result $G = [F]^{nnf}$ of* **NNF**-*transformation can be computed in polynomial time in $|F|$ and produces a formula G in negation normal form such that (i) G is equivalent to F and (ii) $|G| \leq 2{\cdot}|F|$.*

The Structural Transformation

Structural transformation, known also as a *subformula renaming*, is the key part of **CNF**-transformation which allows one to avoid exponentially long **CNF**s. The problem can be demonstrated by the following example:

$$\mathtt{P}_n \;=\; (\mathtt{A}_1 \wedge \mathtt{A}_1') \vee (\mathtt{A}_2 \wedge \mathtt{A}_2') \vee \cdots \vee (\mathtt{A}_n \wedge \mathtt{A}_n'), \qquad n > 0 \tag{3.27}$$

whose *equivalent* **CNF**, consisting of 2^n clauses. This blowup can be avoided by introducing auxiliary propositional "names" $\mathtt{B}_i$ for every conjunct $\mathtt{A}_i \wedge \mathtt{A}_i'$, $1 \leq i \leq n$. Then the the following $2n + 1$ clauses are *conservative over* $\mathtt{P}_n$:

$$\begin{array}{cccc}
\multicolumn{4}{c}{\mathtt{B}_1 \vee \mathtt{B}_2 \vee \cdots \vee \mathtt{B}_n;} \\
\neg\mathtt{B}_1 \vee \mathtt{A}_1; & \neg\mathtt{B}_2 \vee \mathtt{A}_2; & \cdots & \neg\mathtt{B}_n \vee \mathtt{A}_n; \\
\neg\mathtt{B}_1 \vee \mathtt{A}_1'; & \neg\mathtt{B}_2 \vee \mathtt{A}_2'; & \cdots & \neg\mathtt{B}_n \vee \mathtt{A}_n';
\end{array}$$

In Figure 3.9 we defined the structural transformation for a set of formulas $\mathcal{F}$ represented by a general grammar, where B_i, $1 \leq i \leq n$ are first-order formulas for

Figure 3.9 The structural transformation for a recursively defined set of formulas

$$\mathcal{F} ::= B_1 \mid B_2 \mid \cdots \mid B_n \mid R_1[F_1,..,F_{k_1}] \mid R_2[F_1,..,F_{k_2}] \mid \cdots \mid R_m[F_1,..,F_{k_m}]$$

$$[F]^{str} := \mathtt{P}_F \wedge [F]^{def};$$

$$\begin{array}{rcll} [F]^{def} := [B_i]^{def} & = & \forall\overline{x}.(\mathtt{P}_F \rightarrow B_i), & \mid \quad 1 \leq i \leq n, \\ [R_j[F_1,..,F_{k_j}]]^{def} & = & \forall\overline{x}.(\mathtt{P}_F \rightarrow R_j[\mathtt{P}_{F_1},..,\mathtt{P}_{F_{k_j}}]) \wedge [F_1]^{def} \wedge \cdots \wedge [F_{k_j}]^{def} & \mid \quad 1 \leq j \leq m. \end{array}$$

base cases of the definition, and $R_j[F_1,..,F_{k_j}]$, $1 \leq j \leq m$ are recursive constructors of new formulas from the old ones. Hereby we assume that every F_i with $1 \leq i \leq k_j$ occurs positively in $R_j[F_1,..,F_{k_j}]$.[13] Definition (3.26) for first-order formulas in negation normal form is an example of such recursive construction. The function $[F]^{str}$ defined recursively in Figure 3.9 computes the result of the structural transformation for a formula $F \in \mathcal{F}$. Here each $\mathtt{P}_F = \mathtt{p}_F(\overline{x})$ is a fresh *definitional predicate* introduced for F, where $\overline{x} = \mathit{free}[F]$. For example, the function $[\cdot]^{def}$ for the set of first-order formulas in negation normal form defined by (3.26) has the following form:

$$\begin{array}{rcll} [F]^{def} := [A]^{def} & = & \forall\overline{x}.(\mathtt{P}_F \rightarrow A) & \mid \\ [\neg A]^{def} & = & \forall\overline{x}.(\mathtt{P}_F \rightarrow \neg A) & \mid \\ [F_1 \vee F_2]^{def} & = & \forall\overline{x}.(\mathtt{P}_F \rightarrow \mathtt{P}_{F_1} \vee \mathtt{P}_{F_2}) \wedge [F_1]^{def} \wedge [F_2]^{def} & \mid \\ [F_1 \wedge F_2]^{def} & = & \forall\overline{x}.(\mathtt{P}_F \rightarrow \mathtt{P}_{F_1} \wedge \mathtt{P}_{F_2}) \wedge [F_1]^{def} \wedge [F_2]^{def} & \mid \\ [\forall y.F_1]^{def} & = & \forall\overline{x}.(\mathtt{P}_F \rightarrow \forall y.\mathtt{P}_{F_1}) \wedge [F_1]^{def} & \mid \\ [\exists y.F_1]^{def} & = & \forall\overline{x}.(\mathtt{P}_F \rightarrow \exists y.\mathtt{P}_{F_1}) \wedge [F_1]^{def} & . \end{array} \tag{3.28}$$

Remark 3.5.9. Please note that the structural transformation is defined not for a formula, but for its *recursive definition*. A formula might be represented by many different recursive definitions, and consequently might have several different results of the structural transformation. Note also that the structural transformation (3.28) applied to a formula in **NNF** produces a formula in **NNF** (according to our convention $A \rightarrow B$ is a shortcut for $\neg A \vee B$). ◇

Note that the result $[G]^{str}$ of the structural transformation for a formula $G \in \mathcal{F}$ computed according to the definitions in Figure 3.9, can be written as follows:

$$[G]^{str} \;=\; \mathtt{P}_G \wedge \bigwedge_{F=B_i} \forall\overline{x}.(\mathtt{P}_F \rightarrow B_i) \wedge \bigwedge_{F\,=\,R_j[F_1,..,F_{k_j}]} \forall\overline{x}.(\mathtt{P}_F \rightarrow R_j[\mathtt{P}_{F_1},..,\mathtt{P}_{F_{k_j}}]), \tag{3.29}$$

where the conjunctions are taken over subformulas F of G. In particular, the total number of conjuncts is at most $|G|$.

[13] The structural transformation can be also defined for negative occurrences of subformulas, however we do not need this since we put formulas into negation normal form first

Proposition 3.5.10. *Let a formula set $\mathcal{F}$ and a function $[\cdot]^{str}$ be defined like in Figure 3.9. Then the result $H := [G]^{str}$ of structural transformation for $G \in \mathcal{F}$ can be computed in polynomial time in $|G|$ and produces a formula H such that (i) H is conservative over G and (ii) $|H| = O(w{\cdot}|G|)$, where $w := \mathit{width}(G)$.*

Skolemization

In the next step of **CNF**-transformation, the existentially quantified variables of a formula are skolemized. For the purpose of obtaining saturation-based decision procedures we use the standard (outermost) Skolemization. Given a formula F in negation normal (3.26), the result $[F]_o^{sk}$ of applying the *outermost Skolemization* to F is recursively defined in Figure 3.10, where A is an atom, $F_1,\ F_2 \in [\mathcal{FO}]^{nnf}$ and $\mathtt{f}_F(\overline{x})$ is a *Skolem function* introduced for the existentially quantified formula $F = \exists y.F_1$ over its free variables $\overline{x} = \mathit{free}[F]$.

Figure 3.10 Skolemization for first-order formulas in **NNF**

$$
\begin{array}{rcll rcll}
[F]_o^{sk} := [A]_o^{sk} & = & A & | & [F_1 \wedge F_2]_o^{sk} & = & [F_1]_o^{sk} \wedge [F_2]_o^{sk} & | \\
[\neg A]_o^{sk} & = & \neg A & | & [\forall y.F_1]_o^{sk} & = & \forall y.[F_1]_o^{sk} & | \\
[F_1 \vee F_2]_o^{sk} & = & [F_1]_o^{sk} \vee [F_2]_o^{sk} & | & [\exists y.F_1]_o^{sk} & = & [F_1{\cdot}\{y/\mathtt{f}_F(\overline{x})\}]_o^{sk} & .
\end{array}
$$

Proposition 3.5.11. *For any formula $F \in [\mathcal{FO}]^{nnf}$ the result of outermost Skolemization $[F]_o^{sk}$ can be computed in polynomial time in $|F|$ such that the following holds: (i) $[F]_o^{sk}$ is conservative over F and (ii) $|[F]_o^{sk}| \leq (w+1){\cdot}|F|$, where $w = \mathit{width}(F)$.*

Clausification

In the last step of **CNF**-transformation the formula is translated into the **CNF**, by dropping the remaining (universal) quantifiers and distributing conjunctions over disjunctions using the usual distributivity properties:

$$A \vee (B \wedge C) \Rightarrow (A \vee B) \wedge (A \vee C) \quad \text{and} \quad (A \wedge B) \vee C \Rightarrow (A \vee C) \wedge (B \vee C)$$

This step might yield an exponential blowup in the size of formula if **CNF**- transformation is applied naïvely (without the structural transformation). In oder to estimate accurately the computational cost of this transformation, we define it as in Figure 3.11 by a recursive function $[F]^{cnf}$ over quantifier-free first-oder formulas:

$$\mathcal{P} ::= L \mid F_1 \vee F_2 \mid F_1 \wedge F_2\ . \tag{3.30}$$

where L is a literal and $F_i \in \mathcal{P}$, $i = 1, 2$.

Figure 3.11 Clausification for quantifier-free first-order formulas

$$[F]^{cnf} := [F \,|\, \bot]^{cnf}; \qquad \begin{array}{rcll} [F \,|\, C]^{cnf} := [L \,|\, C]^{cnf} & = & C \vee L & | \\ [F_1 \vee L \,|\, C]^{cnf} & = & [F_1 \,|\, C \vee L]^{cnf} & | \\ [F_1 \wedge F_2 \,|\, C]^{cnf} & = & [F_1 \,|\, C]^{cnf} \wedge [F_2 \,|\, C]^{cnf} & | \\ [F_1 \vee (F_2 \wedge F_3) \,|\, C]^{cnf} & = & [F_1 \vee F_2 \,|\, C]^{cnf} \wedge [F_1 \vee F_3 \,|\, C]^{cnf} & | \\ [F_1 \vee (F_2 \vee F_3) \,|\, C]^{cnf} & = & [(F_1 \vee F_2) \vee F_3 \,|\, C]^{cnf}\ . & \end{array}$$

$$\begin{array}{rcll} |F|_{cnf} := |L|_{cnf} & = & 1 & | \\ |F_1 \vee F_2|_{cnf} & = & |F_1|_{cnf} + |F_2|^{\mathbf{1}}_{cnf} + 1 & | \\ |F_1 \wedge F_2|_{cnf} & = & |F_1|_{cnf} + |F_2|_{cnf} + 1\ . & \end{array} \qquad \begin{array}{rcll} |F|^1_{cnf} := |L|^1_{cnf} & = & 1 & | \\ |F_1 \vee F_2|^1_{cnf} & = & |F_1|^1_{cnf} + |F_2|^1_{cnf} + \mathbf{2} & | \\ |F_1 \wedge F_2|^1_{cnf} & = & |F_1|^1_{cnf} + |F_2|^1_{cnf} + 1\ . & \end{array}$$

Proposition 3.5.12. *For any formula $F \in \mathcal{P}$ defined according to* (3.30)*, the result of Clausification $S := [F]^{cnf}$ can be computed in polynomial time in $2^{|F|}$ and produces a formula S in* **CNF** *such that (i) $S \equiv F$ and (ii) $|S| \leq 2^{|F|}$.*

Remark 3.5.13. Although clausification, as stated above, is exponential in the worst case, most translations that we give for fragments are *polynomial.* This is mainly because of the structural transformation step which is applied before clausification. Since the structural transformation produces a formula of the form (3.29), we need to perform clausification only for conjuncts of these formula, which have a fixed form for a fragment. Therefore, every conjunct is translated into **CNF** with a linear overhead in the size and clausification can be computed in polynomial time. ◇

Summing up the results of Propositions 3.5.8, 3.5.10, 3.5.11 and Remark 3.5.13, we can conclude that **CNF**-transformation with a suitable structural transformation can be computed in a polynomial time and produces the result of size $O(w{\cdot}|F|)$, where w is the width of the input formula F.

Chapter 4

Saturation-Based Decision Procedures

In this chapter we describe decision procedures for several fragments of first-order logics: the *guarded fragment*, the *two-variable fragment*, the *monadic fragment*, their combinations and extensions. All these decision procedures are obtained using the *ordered resolution* and *ordered paramodulation* calculi introduced in the previous chapter.

The fact that resolution can be turned into a decision procedure for first-order fragments has been first observed by Joyner Jr. [1976], who formulated several strategies that allow one to decide some prefix-vocabulary classes. For the last thirty years, his method was extended to a variety of other decidable fragments of first-order logic [Tammet, 1990; Fermüller et al., 1993; Fermüller & Salzer, 1993; Bachmair, Ganzinger & Waldmann, 1993*b*; de Nivelle, 1995; Schmidt, 1997; Waldmann, 1997; Hustadt & Schmidt, 1999; Hustadt, 1999; de Nivelle, 2000*a*; Ganzinger & de Nivelle, 1999; Ganzinger, Hustadt, Meyer & Schmidt, 2001; Armando, Ranise & Rusinowitch, 2001; de Nivelle & Pratt-Hartmann, 2001; de Nivelle & de Rijke, 2003; Hustadt et al., 2004].

The principle behind such procedures is very simple: (1) translate a formula from a fragment of interest into a *clause normal form* and (2) demonstrate that resolution *always terminates* on these clauses. This means that a saturation-based prover is guaranteed to produce a correct answer *satisfiable*/*unsatisfiable* for formulas of this fragment in *finite time*. In chapter 2 we have already seen a simple application of this method, where a resolution-based procedure has been defined that decides *subsumption of concepts* in $\mathcal{EL}$.

In order to demonstrate that saturation always terminates for every formula of a fragment, one usually defines a suitable *clause class* for a fragment which (***i***) consists of finitely many clauses for a fixed signature, (***ii***) contains the input clauses and (***iii***) is closed under all inference rules of a calculus. For $\mathcal{EL}$, such clause class was defined by listing *types of clauses* which could appear in a saturation (see Table 2.5 on p.20).

In general, defining a suitable *clause class* which enjoys properties (i) – (iii) is a non-trivial and rather creative task (which can be compared with finding invariants in inductive proofs). Even for relatively simple fragments considered in [Joyner Jr., 1976], the descriptions of suitable clause classes are much more involved than that of DL $\mathcal{EL}$. Hence for fragments that correspond to expressive description logic, this task might be already unfeasible. Therefore the goal of this chapter is not only to give an account of new saturation-based decision procedures (such works are listed above), but also to identify *general strategies and techniques* that support the development of such decision procedures.

The contribution of this chapter can be briefly summarised as follows:

- We introduce a special *scheme notation* which facilitates description of clause classes and inferences between them. Using this notation one can "program" saturation-based decision procedures in tables by listing possible inferences between *clause types*.

- We demonstrate how the *forms* of *input clauses* for fragments can be obtained directly from their *recursive definitions*. These clauses determine the clause class for a fragment.

- We give a *uniform* description of resolution-based decision procedures for the guarded, two-variable and monadic fragments without equality, and show how these fragments and their resolution procedures can be *combined* in a *modular way* (so that the combined procedure remains the same when restricted to each individual fragment).

- We analyse the complexities of our decision procedures and show that they are not only *theoretically optimal*, but also yield the *best known* upper bounds for the considered fragments.

- Finally, we obtain *new* paramodulation-based decision procedures for extensions of the *guarded fragment* with *equality*, *constants*, *functionality* and *counting*. These fragments have important relationship with many well-known expressive description logics.

4.1 Decision Procedures Based on Ordered Resolution

4.1.1 Deciding the Guarded Fragment without Equality

de Nivelle [1998] was the first to formulate a resolution-based decision procedure for the guarded fragment without equality [see also the extended version of this paper: de Nivelle & de Rijke, 2003]. This decision procedure uses a refinement of resolution by non-liftable orders, completeness of which is demonstrated by *resolution games* [de Nivelle, 1995]. In fact, the procedure given in this paper decides a much larger clause class, than is actually obtained from the translation of guarded formulas (given in this paper). A more concise clause class for the guarded fragment has been defined in [Ganzinger & de Nivelle, 1999], where a paramodulation-based decision procedure for the guarded fragment *with equality* has been found. The smaller clause class makes it possible to employ liftable orders, which allows one to use any "off-the-shelf" saturation-based theorem prover to implement the decision procedure.

In this section we define a small clause class that captures the guarded fragment without equality, and gives rise to many decision procedures for extensions of the guarded fragment that will be presented in subsequent sections. Our aim is not only to show decidability of these fragments by standard resolution and paramodulation, but also to obtain sharp complexity bounds for these fragments. We will show that for the guarded, two-variable and monadic fragments, our procedures have the best known complexity.

The plan for the rest of this section is the following. First, we sketch a decision procedure for the class of guarded clauses defined in [de Nivelle, 1998; de Nivelle & de Rijke, 2003]. After that, we review a **CNF**-transformation procedure for the guarded formulas and define a smaller clause class, that captures the result of the translation. Finally, we present a decision procedure for the defined clause class by ordered resolution and estimate its complexity.

Guarded clauses and non-liftable orders

A clause class defined by de Nivelle [1998] for the guarded fragment inherits certain features from the class $\mathcal{E}^+$, that has been introduced by Tammet [1990] and later studied in [Fermüller et al., 1993; de Nivelle, 1995, 2000*a*]. $\mathcal{E}^+$ is defined as a class of so-called *variable uniform* clauses. A clause C is *variable uniform* if (***i***) all literals in C are *weakly covering* (recall Definition 3.1.13 and terminology from 3.1.4) and (***ii***) for each two literals L_1 and L_2 of C, either $\mathit{vars}[L_1] = \mathit{vars}[L_2]$, or $\mathit{vars}[L_1] \cap \mathit{vars}[L_2] = \{\}$.

The following definition is a slightly modified definition of guarded clauses from [de Nivelle & de Rijke, 2003]:

Definition 4.1.1. A clause C is *guarded* *iff*
(***i***) C is a weakly covering clause (see Definition 3.1.13), and
(***ii***) if C is not ground, then there is a negative literal $\neg A$ in C (called a *guard*) that contains all variables of C.[1] ◊

For example, the following clauses are guarded, where the appropriate guards are underlined:

$\mathtt{a}(\mathtt{c},\mathtt{g}(\mathtt{f}(\mathtt{c}),\mathtt{c}))$; $\neg\mathtt{b}(\mathtt{x}) \vee \underline{\neg\mathtt{a}(\mathtt{x},\mathtt{y})} \vee \mathtt{a}(\mathtt{f}(\mathtt{c}),\mathtt{g}(\mathtt{x},\mathtt{y}))$;
$\underline{\neg\mathtt{a}(\mathtt{x},\mathtt{f}(\mathtt{c}))} \vee \neg\mathtt{a}(\mathtt{c},\mathtt{f}(\mathtt{f}(\mathtt{x}))) \vee \neg\mathtt{a}(\mathtt{c},\mathtt{f}(\mathtt{c}))$.

The second literal of the last clause can also be selected as a guard, but not the last literal. The following clauses are not guarded:

$\neg\mathtt{b}(\mathtt{x}) \vee \neg\mathtt{a}(\mathtt{x},\mathtt{y}) \vee \mathtt{a}(\mathtt{f}(\mathtt{c}),\mathtt{g}(\mathtt{x},\mathtt{c}))$ - *the last literal is not weakly covering;*
$\neg\mathtt{b}(\mathtt{x}) \vee \neg\mathtt{b}(\mathtt{y}) \vee \mathtt{a}(\mathtt{x},\mathtt{y})$ - *the clause is not ground but there is no guard;*

The main idea behind the decision procedure for the class of guarded clauses by resolution (as well as for the class $\mathcal{E}^+$), is to force a procedure to perform resolution inferences only on literals such that: (**1**) *a literal contains all variables of the clause* and (**2**) *a literal has the maximal (variable) depth in the clause.* The first condition usually guarantees that the number of different variables in a resolvent do not exceed the maximal number of different variables in every clause it is derived from. The second conditions allows one to establish a bound on (variable) depth of the derived clause. These conditions together insure that all derived clauses have a bounded size, therefore, only finitely many of those can be generated during saturation.

In order to fulfil conditions (1) and (2) above for guarded clauses, de Nivelle [1998] introduces the following ordering $\succ$ on *non-ground* literals:

$$L_1 \succ L_2 \quad \textit{if} \quad \begin{array}{l} (\boldsymbol{a})\ \mathit{vars}[L_2] \subsetneq \mathit{vars}[L_1],\ \textit{or} \\ (\boldsymbol{b})\ \mathit{vars}[L_1] = \mathit{vars}[L_2],\ \text{but}\ \mathit{vardepth}(L_1) > \mathit{vardepth}(L_2). \end{array} \tag{4.1}$$

The ordering $\succ$ satisfying these properties is non-liftable (recall the terminology from subsection 3.5.2). Both conditions of the ordering (4.1) violate liftability: both (a) $\mathtt{a}(x) \succ \mathtt{a}'(\mathtt{c})$; $\mathtt{a}'(x) \succ \mathtt{a}(\mathtt{c})$ and (b) $\mathtt{b}(\mathtt{f}(x),\mathtt{c}) \succ \mathtt{b}'(\mathtt{f}(\mathtt{c}),x)$; $\mathtt{b}'(\mathtt{f}(x),\mathtt{c}) \succ \mathtt{b}(\mathtt{f}(\mathtt{c}),x)$ yield conflicts under the substitution $\{x/\mathtt{c}\}$, if $\mathtt{a} \neq \mathtt{a}'$, $\mathtt{b} \neq \mathtt{b}'$. Nonetheless, de Nivelle [1998] proves that ordered resolution based on an ordering $\succ$ fulfilling the conditions above, remains complete for guarded clauses:

[1] this case contained more restrictions, which, however are not necessary for the procedure

Theorem 4.1.2. [de Nivelle, 1998] *Let N be a set of guarded clauses and N' be a saturation of N under ordered resolution based on an order* (4.1)*, applied a-priori. Then*
(**1**) *all clauses in N' are guarded;*
(**2**) *N' is satisfiable iff N is satisfiable;*
(**3**) *$depth(N') \leq vardepth(N) + depth(N)$.*

Theorem 4.1.2 implies that a set of the guarded clauses can be decided by ordered resolution based on non-liftable orders. There are some disadvantages of using non-liftable orders in resolution. The important one, is that non-liftable orders are not fully compatible with redundancy elimination techniques, in particular, with Tautology Deletion.

Satisfiability for guarded clauses can probably be decided using a liftable order applied *a-posteriori* (i.e. when ordering conditions are verified *after* a rule is applied), similar as it has been done for the class $\mathcal{E}^+$ in [de Nivelle, 2000*a*] (but the procedure yields a larger bound on a clause depth). However, for deciding the guarded fragment, many problems can be avoided by considering a smaller clause class.

Clause Schemes

Before we start describing our resolution-based decision procedure for the guarded fragment, we introduce a special *scheme notation* to facilitate description of clauses and inferences between them. We augment the usual first-oder notation with a constructor of the type:

$$< \text{top symbols} > (< \text{arguments} >)$$

which represents a term, literal or a clause having certain top symbols (functional, predicate symbol, negation of a predicate symbol, etc.) and certain sequence of arguments. For example, $(p|\neg p)(\overline{x}, \overline{y})$ stands for either $p(\overline{x}, \overline{y})$ or $\neg p(\overline{x}, \overline{y})$. Here the arguments of both expressions $(\overline{x}, \overline{y})$ are composed from two sequences of (not nececerily disjoint) variable-vectors $\overline{x}$ and $\overline{y}$. If we want to describe a similar expression whose arguments are *from* either $\overline{x}$ or $\overline{y}$, we write $(p|\neg p)[\overline{x}, \overline{y}]$. This form means neither that all variables from $\overline{x}$, $\overline{y}$ occur as arguments, nor that they follow in some specific order. If we want to additionally express, say, that *all* variables from $\overline{x}$ occur as arguments of the expression, we write $(p|\neg p)[!\overline{x}, \overline{y}]$.

We can form more complicated expressions by nesting our scheme notation. For example, we can write $\{\vee l\}[\overline{x}, !\hat{f}(\overline{x})]$ to denote a clause consisting of literals whose arguments are either from $\overline{x}$, or of form $\hat{f}(\overline{x})$, and at least one such functional term is present. The "hat" on top of f means that there are possibly several occurrences of functional terms of this form for *different* functional symbols. If we had written $\{\vee l\}[\overline{x}, !f(\overline{x})]$, this would mean that all functional symbols in the clause must be the same. Symbols f and $\hat{f}$ are called *parameters* (for functional symbols). The first parameter is *free*, i.e., all occurrences of this parameter must correspond to the *same* functional symbol. The second parameter is *bounded*, meaning that different occurrences of this parameter may correspond to *different* functional symbols. We may also define new parameters using the old ones. For example, the *literal parameter l* in the above scheme can be defined as $l := p|\neg p$. Similarly we can introduce a *clause parameter α* $:= \{\vee l\}$, and write the above scheme as $\alpha[\overline{x}, !\hat{f}(\overline{x})]$ or $\hat{\alpha}[\overline{x}, !\hat{f}(\overline{x})]$.

Please see Appendix B containing a comprehensive account on our scheme language, including formal definitions and numerous examples. However, we also try to give comments for the scheme expressions that we use, so their meanings should be clear enough from what we have just said.

A clause class for the guarded fragment

To obtain a clause class for the guarded fragment, we apply the general **CNF**-transformation procedure that we have described in subsection 3.5.7. We start with a recursive definition (3.17) for the guarded formulas given in subsection 3.3.3:

$$\mathcal{GF} \ ::= \ A \mid \neg F_1 \mid F_1 \wedge F_2 \mid \forall \overline{y}.[G \rightarrow F_1] \ . \tag{4.2}$$

where A is an atom containing no functional terms[2], F_i with $i = 1, 2$ are guarded formulas and G is an atom, called a *guard* containing all free variables of the corresponding subformula F_1.

After applying the negation normal form transformation (3.25) for formulas of form (4.2), we obtain a fragment that can be characterised as follows:

$$[\mathcal{GF}]^{nnf} \ ::= A \mid \neg A \mid F_1 \wedge F_2 \mid F_1 \vee F_2 \mid \forall \overline{y}.[G \rightarrow F_1] \mid \exists \overline{y}.[G \wedge F_1] \ . \tag{4.3}$$

As has been shown in 3.5.7 (see Proposition 3.5.8), **NNF**-transformation can be computed in polynomial time and produces a linear result in the size of the input formula.

Now it's time for the structural transformation, which we apply according to the general procedure (3.9). For any $F \in [\mathcal{GF}]^{nnf}$ defined in (4.3), the result of structural transformation is $[F]^{str}_g := \mathtt{P}_F \wedge [F]^{def}_g$, where the function $[\cdot]^{def}_g$ is defined

[2] A decision procedure for the guarded fragment with constants will be considered in subsection 4.3.2

below:

$$\begin{aligned}
[F]_g^{def} := [A]_g^{def} &= \forall\overline{x}.(\mathtt{P}_F \rightarrow A) & \mid\\
[\neg A]_g^{def} &= \forall\overline{x}.(\mathtt{P}_F \rightarrow \neg A) & \mid\\
[F_1 \wedge F_2]_g^{def} &= \forall\overline{x}.(\mathtt{P}_F \rightarrow \mathtt{P}_{F_1} \wedge \mathtt{P}_{F_2}) \wedge [F_1]_g^{def} \wedge [F_2]_g^{def} & \mid\\
[F_1 \vee F_2]_g^{def} &= \forall\overline{x}.(\mathtt{P}_F \rightarrow \mathtt{P}_{F_1} \vee \mathtt{P}_{F_2}) \wedge [F_1]_g^{def} \wedge [F_2]_g^{def} & \mid\\
[\forall\overline{x}.[G \rightarrow F_1]]_g^{def} &= \forall\overline{x}.(\mathtt{P}_F \rightarrow \forall\overline{y}.(G \rightarrow \mathtt{P}_{F_1})) \wedge [F_1]_g^{def} & \mid\\
[\exists\overline{x}.[G \wedge F_1]]_g^{def} &= \forall\overline{x}.(\mathtt{P}_F \rightarrow \exists\overline{y}.(G \wedge \mathtt{P}_{F_1})) \wedge [F_1]_g^{def}\ .
\end{aligned} \tag{4.4}$$

Hereby $\mathtt{P}_F = \mathtt{p}_F(\overline{x})$ is, as usual, a *definitional predicate* for a guarded subformula F over $\overline{x} = \mathit{free}[F]$. According to (3.29), the result of the structural transformation is a conjunction of formulas whose types are shown on Table 4.1 in the left column, where we have additionally applied the existential closure. Here we partially use our scheme notation: $\overline{x}$, $\overline{y}$ denote sequences of variables; $\mathtt{p}_F(\overline{x})$ denotes an atom whose sequence of arguments is $\overline{x}$; an atom $a[\overline{x}]$ ($a[\overline{x}, \overline{y}]$) contains only variables from $\overline{x}$ (resp. from $\overline{x}$ and from $\overline{y}$) and an atom $a[!\overline{x}]$ ($a[!\overline{x}, !\overline{y}]$), in addition, contains at least one occurrence for every variable in $\overline{x}$ (resp. in $\overline{x}$ and in $\overline{y}$).

Table 4.1 Types of clauses resulted form **CNF** transformation for guarded formulas

Type of a conjunct	$\dashrightarrow [\cdot]^{sk}, [\cdot]^{cnf} \dashrightarrow$	**Type of a clause**	**(Nr)**
$\exists\overline{x}.\mathtt{p}_F(\overline{x})$	$\dashrightarrow$	$\mathtt{p}_F(\mathtt{c}_{sk})$	(1)
$\forall\overline{x}.(\mathtt{p}_F(\overline{x}) \rightarrow a[!\overline{x}])$	$\dashrightarrow$	$\neg\mathtt{p}_F(\overline{x}) \vee a[!\overline{x}]$	(2)
$\forall\overline{x}.(\mathtt{p}_F(\overline{x}) \rightarrow \neg a[!\overline{x}])$	$\dashrightarrow$	$\neg\mathtt{p}_F(\overline{x}) \vee \neg a[!\overline{x}]$	(3)
$\forall\overline{x}.(\mathtt{p}_F(\overline{x}) \rightarrow (\mathtt{p}_{F_1}[\overline{x}] \wedge \mathtt{p}_{F_2}[\overline{x}]))$	$\dashrightarrow$	$\neg\mathtt{p}_F(\overline{x}) \vee \mathtt{p}_{F_1}[\overline{x}]$	(4)
		$\neg\mathtt{p}_F(\overline{x}) \vee \mathtt{p}_{F_2}[\overline{x}]$	(4)
$\forall\overline{x}.(\mathtt{p}_F(\overline{x}) \rightarrow (\mathtt{p}_{F_1}[\overline{x}] \vee \mathtt{p}_{F_2}[\overline{x}]))$	$\dashrightarrow$	$\neg\mathtt{p}_F(\overline{x}) \vee \mathtt{p}_{F_1}[\overline{x}] \vee \mathtt{p}_{F_2}[\overline{x}]$	(5)
$\forall\overline{x}.(\mathtt{p}_F(\overline{x}) \rightarrow \forall\overline{y}.(g[!\overline{x}, !\overline{y}] \rightarrow \mathtt{p}_{F_1}[\overline{x}, \overline{y}]))$	$\dashrightarrow$	$\neg g[!\overline{x}, !\overline{y}] \vee \neg\mathtt{p}_F(\overline{x}) \vee \mathtt{p}_{F_1}[\overline{x}, \overline{y}]$	(6)
$\forall\overline{x}.(\mathtt{p}_F(\overline{x}) \rightarrow \exists\overline{y}.(g[!\overline{x}, !\overline{y}] \wedge \mathtt{p}_{F_1}[\overline{x}, \overline{y}]))$	$\dashrightarrow$	$\neg\mathtt{p}_F(\overline{x}) \vee g[!\overline{x}, !\overline{\mathtt{f}}_{sk}(\overline{x})]$	(7)
		$\neg\mathtt{p}_F(\overline{x}) \vee \mathtt{p}_{F_1}[\overline{x}, \overline{\mathtt{f}}_{sk}(\overline{x})]$	(8)

Applying (the outermost) Skolemization and Clausification to the formulas of these types, we obtain clauses of types (1) – (8) shown in the right column of Table 4.1. Note, that all clauses (1) – (8) are *guarded* according to Definition 4.1.1: ***(i)*** All clauses are *weakly covering* (and even *covering*!), since all functional terms are either Skolem constants $\mathtt{c}_{sk}$ in ground clauses, or Skolem functions $\mathtt{f}_{sk}(\overline{x})$ containing all variables $\overline{x}$ of the clause in which they occur. ***(ii)*** All non-ground clauses have a guard containing all variables of a clause: the negated *definitional predicate* $\neg\mathtt{p}_F(\overline{x})$ acts as guard for the clauses (2) – (5) and (7) – (8); for the clause (6) the guard is the literal $\neg g[!\overline{x}, !\overline{y}]$.

The clauses (1) – (8) in Table 4.1 are guarded clauses of a very specific form:

1. *All clauses are* simple, *i.e. they contain only shallow subterms (or equivalently, their depth is not greater than* 3*);*
2. *All guards in clauses contain no functional subterms (i.e. guards contain only variables);*
3. *All functional subterms in clauses have the same sequence of arguments, which may be only variables (and all variables that occur in a clause should be there).*

The properties 1-2 have been taken into account in [Ganzinger & de Nivelle, 1999], where a more restricted class of clauses (also called guarded clauses) has been introduced. We restrict this class even further by additionally imposing condition 3. Additional conditions allow us to obtain a smaller complexity bound for the guarded fragment. Note, that property 3 implies that ground clauses may contain only constants as functional subterms.

A subclass (**G**) of guarded clauses having properties 1-3 is defined using *clause schemes* in Table 4.2. Here $\hat{c}$ is a bounded constant parameter that ranges over the set of all constants; $\hat{f}$ is a bounded functional parameter that ranges over the set of all functional symbols (including constants); p is a predicate parameter that ranges over the set of all predicate symbols and α is the correspondent clause parameter for p (for the detail of these notations please refer to Appendix B).

Table 4.2 A clause class for the guarded fragment without equality

	Clause scheme	***Description***
	1 $\hat{\alpha}[\hat{c}]$	*a clause containing only constants;*
(**G**):	2 $\neg\hat{p}[!\overline{x}] \vee \hat{\alpha}[\hat{f}(\overline{x}), \overline{x}]$	*a clause for which there is an enumeration of its variables* $\overline{x}$ *(possibly with repetitions) such that there is a negative literal - guard* $\neg\hat{p}[!\overline{x}]$*, containing all variables from* $\overline{x}$*, and such that all functional terms of the clause have arguments* $\overline{x}$*.*
	where $l := p \mid \neg p$; $\alpha := \vee\{l\}$	

Proposition 4.1.3. *Let* $F \in \mathcal{GF}$ *be a guarded formula,* $\boldsymbol{n} = |F|$ *and* $\boldsymbol{w} = width(F)$*. Then* **CNF***-transformation for* F *can be computed in polynomial time in* $\boldsymbol{n}$ *and produces at most* $2\boldsymbol{n}$ *clauses with at most* 3 *literals, which belong to* (**G**) *and have at most* $\boldsymbol{w}$ *different variables.*

Proof. As has been argued in 3.5.7, the structural transformation for the guarded fragment can be computed in polynomial time in $\boldsymbol{n}$ and produces at most $\boldsymbol{n}$ elementary conjuncts. Each conjunct gives rise to at most 2 clauses (see Table 4.1). Each of these clauses consists of at most 3 literals. It remains to notice that the number of different variables in each clause is bounded by the maximal number of different variables in every guard of F which is at most $\boldsymbol{w}$. ⧉

Saturation of the clause set

Now we show that the clause class (**G**) in Table 4.2 is closed under inferences of ordered resolution with selection $\mathcal{OR}^{\succ}_{Sel}$ (see System 3) with *eager elimination of duplicate literals and tautology deletion*, based on a quite general class of orderings. We need to restrict an order $\succ$ on literals in such a way, that eligible literals (i.e., the literals on which clauses are resolved) are of the maximal depth and contain all variables of the clause (recall two basic conditions for decidability by resolution formulated in the beginning of this chapter). Hopefully, we can achieve both of these goals for clauses from (**G**) using *liftable* orders.

Definition 4.1.4. We say that an ordering $\succ$ on ground literals is *simple iff* $l(s_1,..,s_n) \succ k(t_1,..,t_m)$ whenever there exists i with $1 \leq i \leq n$ such that $s_i \rhd t_j$ for all j with $1 \leq j \leq m$. ◇◇

Simple orders do exist. For example, any *LPO* order $\succ_{lpo}$ (see Definition 3.1.5) based on a precedence $\gg$, in which $f \gg a$ for any non-constant functional symbol f and a predicate symbol a, has the above property. However, is not possible to find a simple *KBO* ordering (see Example 3.1.8). Within this section we assume that $\mathcal{OR}^{\succ}_{Sel}$ is parametrized by some simple order $\succ$.

Having a simple order $\succ$, we now can be sure that a maximal literal in a functional clause form (**G**) is also functional. Indeed, it is not possible that $k[\overline{x}] \succ l[!\hat{f}(\overline{x}), \overline{x}]$ (this notation means that the last literal *contains* an argument $f(\overline{x})$), since $x \lhd f(\overline{x})$ for every $x \in \overline{x}$. Thus, maximal literals in *functional* clauses from Table 4.2 are of the maximal depth and contain all variables of the clause.

Non-functional clauses from (**G**) have form $\neg\hat{p}[!\overline{x}] \vee \hat{\alpha}[\overline{x}]$. Unfortunately there is no guarantee that for a clause of this form, a maximal literal contains all of its variables. Luckily, all such clauses must have a guard, which is a negative literal that *contains all variables* of a clause. So, a *selection function* can be used for making the guard literal eligible. We define a selection function *Sel* for clauses of form $\neg\hat{p}[!\overline{x}] \vee \hat{\alpha}[\overline{x}]$, to select a guard $\boldsymbol{\neg\hat{p}[!\overline{x}]}^{\sharp}$ of the clause (selected literals are indicated by symbol $\sharp$).

The case analysis of possible $\mathcal{OR}^{\succ}_{Sel}$ inferences from clauses of (**G**) is summarised in Table 4.3. We will use tables of this kind to demonstrate closure of different clause classes under inferences of appropriate calculi. Below we explain in details the meaning of such a table.

Table 4.3 is organized as follows. The clause schemes from (**G**) are split on several other subschemes distinguished by different forms of eligible literals (on the first level) and applicable inference rules (on the second level). This means that first, possible types of maximal and selected literals are identified. For example, a functional literal of form $\boldsymbol{l[!\hat{f}(\overline{x}), \overline{x}]}$ might be maximal in a clause scheme 2, and

Table 4.3 Possible inferences between clauses for the guarded fragment

1	$\hat{\alpha}[\hat{c}]$	
1.1	$\hat{\alpha}[\hat{c}] \vee \boldsymbol{l[\hat{c}]}$	
1.1.1	$\hat{\alpha}[\hat{c}] \vee \underline{\boldsymbol{p[\hat{c}]}}^{\star}$	$:\texttt{OR}.1$
1.1.2	$\hat{\alpha}[\hat{c}] \vee \neg\underline{\boldsymbol{p[\hat{c}]}}$	$:\texttt{OR}.2$
1.1.3	$\hat{\alpha}[\hat{c}] \vee \underline{p[\hat{c}]} \vee \underline{\boldsymbol{p[\hat{c}]}}$	$:[\![\,\texttt{OF}\,]\!]$
	$\texttt{OR}[1.1.1; 1.1.2]\!: \hat{\alpha}[\hat{c}]$	$:1$
$\perp$	$\square$	

2	$\neg\hat{p}[!\overline{x}] \vee \hat{\alpha}[\hat{f}(\overline{x}), \overline{x}]$	
2.1	$\neg\hat{p}[!\overline{x}] \vee \hat{\alpha}[\hat{f}(\overline{x}), \overline{x}] \vee \boldsymbol{\hat{l}[!\hat{f}(\overline{x}), \overline{x}]}$	
2.1.1	$\neg\hat{p}[!\overline{x}] \vee \hat{\alpha}[\hat{f}(\overline{x}), \overline{x}] \vee \underline{\boldsymbol{p[!\hat{f}(\overline{x}), \overline{x}]}}^{\star}$	$:\texttt{OR}.1$
2.1.2	$\neg\hat{p}[!\overline{x}] \vee \hat{\alpha}[\hat{f}(\overline{x}), \overline{x}] \vee \neg\underline{\boldsymbol{p[!\hat{f}(\overline{x}), \overline{x}]}}$	$:\texttt{OR}.2$
2.1.3	$\neg\hat{p}[!\overline{x}] \vee \hat{\alpha}[\hat{f}(\overline{x}), \overline{x}] \vee \underline{p[\hat{f}(\overline{x}), \overline{x}]} \vee \underline{\boldsymbol{p[!\hat{f}(\overline{x}), \overline{x}]}}$	$:\texttt{OF}$
	$\texttt{OR}[2.1.1; 2.1.2]\!: \neg\hat{p}[!\overline{x}] \vee \hat{\alpha}[\hat{f}(\overline{x}), \overline{x}]$	$:2$
	$\texttt{OF}[2.1.3]\ : \neg\hat{p}[!\overline{x}] \vee \hat{\alpha}[\hat{f}(\overline{x}), \overline{x}] \vee p[!\hat{f}(\overline{x}), \overline{x}]$	$:2$
2.2	$\neg\boldsymbol{\hat{p}[!\overline{x}]}^{\sharp} \vee \hat{\alpha}[\overline{x}]$	$:\mathit{Sel}$
2.2.1	$\neg\underline{\boldsymbol{\hat{p}[!\overline{x}]}} \vee \hat{\alpha}[\overline{x}]$	$:\texttt{OR}.2$
	$\texttt{OR}[1.1.1; 2.2.1]\!: \hat{\alpha}[\hat{c}]$	$:1$
	$\texttt{OR}[2.1.1; 2.2.1]\!: \neg\hat{p}[!\overline{x}] \vee \hat{\alpha}[\hat{f}(\overline{x}), \overline{x}]$	$:2$

this is what the clause scheme 2.1 represents. Or, otherwise a clause should be non-functional, and therefore, the selected guard is an eligible literal. This case is represented by the clause scheme 2.2, where we have marked the selected literal by $\sharp$ and indicated the use of selection function to the right of the clause scheme.

Once the form of an eligible literal is identified, a clause scheme is matched against possible premises of inference rules of the calculus (for $\mathcal{OR}^{\succ}_{Sel}$ see System 3 on p. 79). The case 2.1.1 represents a situation when a maximal literal is positive: $\boldsymbol{p[!\hat{f}(\overline{x}), \overline{x}]}$, thus a clause may be used as a first premise of the Ordered Resolution rule. This is indicated by placing $\texttt{OR}.1$ to the right of this case. The schemes 2.1.2 and 2.1.3 represent cases when a clause can be used as a second premise of the Ordered Resolution rule ($\texttt{OR}.2$), and, respectively, as a premise of the Ordered Factoring rule ($\texttt{OF}$). A clause of form 2.2 can be only used as a second premise of Ordered Resolution rule ($\texttt{OR}.2$), since the selected literal is negative. This is indicated in the case 2.2.1. All cases should be exhaustive, i.e. all possibilities of applying inference rules should be enumerated.

After all possibilities of using a clause in inference rules are identified, we enumerate inferences between them. A clause of form 2.1.1 can be resolved with a clause of form 2.1.2. This is indicated in Table 4.3 in a line starting with $\texttt{OR}[2.1.1; 2.1.2]$. To characterize the result of this inference, consider the situation in more details. We have two clauses of the following forms:

2.1.1 $\neg\hat{p}[!\overline{x}] \vee \hat{\alpha}[\hat{f}(\overline{x}), \overline{x}] \vee \underline{\boldsymbol{p[!\hat{f}(\overline{x}), \overline{x}]}}^{\star}$ $:\texttt{OR}.1$
2.1.2 $\neg\hat{p}[!\overline{y}] \vee \hat{\alpha}[\hat{f}(\overline{y}), \overline{y}] \vee \underline{\boldsymbol{\neg p[!\hat{f}(\overline{y}), \overline{y}]}}$ $:\texttt{OR}.2$

which is the same as in Table 4.3, except that we have renamed the variables of clauses apart. Let $\sigma = mgu(p[!\hat{f}(\overline{x}), \overline{x}], p[!\hat{f}(\overline{y}), \overline{y}])$ be a unifier used in the inference $\texttt{OR}[2.1.1; 2.1.2]$. Since the unified expressions are covering and of equal depth (both contain at least one functional term), by Lemma 3.1.14 (see 3.1.4), σ is atomic for

$\overline{x}$ and for $\overline{y}$. Since there are no constants in expressions, $Ran(\sigma)$ consists only of variables. Moreover, σ maps $\overline{x}$ and $\overline{y}$ to the *same* sequence of variables, since a functional subterm $f(\overline{x})$ of the first expression must be unified with a functional subterm $f(\overline{y})$ of the second expressions (it cannot be unified with a variable, since σ is atomic). So, $\sigma = \{\overline{x}/\overline{z}, \overline{y}/\overline{z}\}$ for some sequence of variables $\overline{z}$, and the result of the inference is:

$$\begin{array}{lll} \mathtt{OR}[2.1.1; 2.1.2]\colon \neg\hat{p}[!\overline{z}] \vee \hat{\alpha}[\hat{f}(\overline{z}), \overline{z}] \vee \neg\hat{p}[!\overline{z}] \vee \hat{\alpha}[\hat{f}(\overline{z}), \overline{z}] & & \\ \Rightarrow \qquad\qquad\quad\ \colon \neg\hat{p}[!\overline{x}] \vee \hat{\alpha}[\hat{f}(\overline{x}), \overline{x}] & & \colon 2 \end{array}$$

One can see that the conclusion of the inference $\mathtt{OR}[2.1.1; 2.1.2]$ is always an instance of scheme 2, which is indicated to the right of the result. Similar analysis allows one to find results for other inferences using Lemma 3.1.14.

The inference rule for the case 1.1.3 is enclosed in brackets: $[\![\, \mathtt{OF} \,]\!]$, since the application of Ordered Factoring rule is redundant: if two literals of a ground clause can be factored, then they are equal and the clause is *simplified* using Elimination of Duplicate Literals, which is always applied *eagerly* (see subsection 3.5.6). We will indicate redundant clauses and redundant rules by enclosing them in brackets $[\![\, .. \,]\!]$. The case $\perp$ corresponds to the empty clause $\square$.

The case analysis in Table 4.3 is complete which demonstrates that the clause class $(\mathbf{G})$ is closed under inferences of $\mathcal{OR}^{\succ}_{Sel}$. This proves the following lemma:

Lemma 4.1.5 (Closure of $(\mathbf{G})$)**.** *Let N be a set of clauses of form $(\mathbf{G})$ and N' be obtained by saturation of N in $\mathcal{OR}^{\succ}_{Sel}$ based on a simple order $\succ$ and a selection function Sel that selects a guard in non-functional guarded clauses. Let $\boldsymbol{w}$ be the maximal number of different variables in a clause from N. Then all clauses from N' belong to $(\mathbf{G})$ and have at most $\boldsymbol{w}$ different variables.*

Lemma 4.1.5 implies that the ordered resolution calculus $\mathcal{OR}^{\succ}_{Sel}$ can be used as a decision procedure for the clause class $(\mathbf{G})$ and, through Proposition 4.1.3, for the guarded fragment $\mathcal{GF}$, since at most finitely many *normalised* clauses from $(\mathbf{G})$ can be generated for a given formula $F \in \mathcal{GF}$ (all these clauses are over a fixed signature, have bounded number of variables and are *without duplicate literals*). In Appendix C.1.1 we have proved that the running time t of our procedure can be bounded by:

$$\boxed{\; t \;\leq\; 2^{\boldsymbol{n} \cdot 2^{\boldsymbol{w} \cdot (\log \boldsymbol{w} + \epsilon)}} \;}$$

where $\boldsymbol{n} := |F|$ is the size of the input guarded formula F, $\boldsymbol{w} := width(F)$ is its width, and ϵ is some constant. A similar complexity bound: $\boldsymbol{t} \leq 2^{\boldsymbol{n} \cdot 2^{\boldsymbol{a} \cdot (\log \boldsymbol{w} + \epsilon)}}$ (where $\boldsymbol{a}$ is the maximal arity of atoms), has been obtained by Grädel [1999] using model-theoretic analysis. Our procedure yields an improved complexity bound over those given in [de Nivelle & de Rijke, 2003]: $\boldsymbol{t} \leq 2^{2^{\boldsymbol{a}^{v} \cdot (\log \boldsymbol{n} + \epsilon)}}$ (where v is the maximal *variable depth* of the initial guarded clauses), and in [Ganzinger & de Nivelle, 1999]: $\boldsymbol{t} \leq 2^{2^{(\boldsymbol{a}^2+\boldsymbol{a}+1)\cdot(\log \boldsymbol{n}+\epsilon)}}$, because we consider a much smaller clause class and estimate complexity more precisely. As a consequence, the following theorem is shown:

Theorem 4.1.6. *There is a resolution-based decision procedure for the guarded fragment without equality $\mathcal{GF}$ that runs in 2EXPTIME. This procedure decides the bounded-variable guarded fragment $\mathcal{GF}^k$ in EXPTIME.*

Note 4.1.7. *Decidability of the guarded fragment has been demonstrated by Andréka et al. [1996] using mosaic techniques known from modal logics. An exponential space alternating decision procedure for guarded formulas has been described by Grädel [1999] yielding 2EXPTIME upper complexity bound for $\mathcal{GF}$. He also noted that a specialisation of this procedure for $\mathcal{GF}^k$ has a lower complexity, namely EXPTIME. This paper also gives a matching upper complexity bound for $\mathcal{GF}$ by reduction from alternating Turing machines with exponential tape, establishing thereby its 2EXPTIME-completes.*

The first resolution-based decision procedure for the guarded fragment has been proposed in [de Nivelle, 1998], which later has been extended for equality [Ganzinger & de Nivelle, 1999]. In the last paper, complexity of the decision procedure is measured concluding that it is theoretically optimal. Similar complexity estimations have been carried out later in [de Nivelle & de Rijke, 2003], where the procedure was also extended for the loosely guarded fragment. Hence the first part of Theorem 4.1.6 should be attributed to these papers. Specialisation of these decision procedures for formulas with bounded number of variables has not been considered in these papers, although EXPTIME upper bound follows from provided calculations for guarded formulas whose predicate and functional symbols have bounded arity – see the estimations in Appendix C.1.1. ◈

4.1.2 Deciding the Two-Variable Fragment without Equality

In this section we describe a resolution-based procedure for the two-variable fragment of first-order logic by essentially repeating the same steps as for the guarded fragment.

Although decidability of the two-variable fragment by resolution has been explicitly stated in [de Nivelle, 2000*b*], this fact follows from previous works on resolution decision procedures, since, in particular, clauses resulted from **CNF**-transformation of two-variable formulas in Scott normal form (3.16) (see p. 74), fall into classes $\mathcal{S}^+$, $\mathcal{E}^+$, and Maslov's class K, for which resolution based decision procedures were studied in [Tammet, 1990; Fermüller et al., 1993; de Nivelle, 1995; Hustadt & Schmidt, 1999]. Moreover, modal logic $\mathbf{K_m}(\cap, \cup, \neg, ^{\smile})$, (a syntactical variants of

$\mathcal{ALC}(^-, \sqcap, \neg)$, see subsection 3.2.2) is essentially the two-variable fragment, and is also known to be decidable by resolution (see [Hustadt, 1999; Schmidt & Hustadt, 2003]).

In this section, we demonstrate how a resolution decision procedure for the two-variable fragment can be obtained from its recursive definition by performing similar steps as for the guarded fragment. We also show that this procedure has the best known complexity.

A clause class for the two-variable fragment

We start with a recursive definition (3.15) for two-variable formulas that was given in subsection 3.3.2:

$$\mathcal{FO}^2 \ ::= \ A[x,y] \mid \neg T_1 \mid T_1[x,y] \wedge T_2[x,y] \mid \forall y.T_1[x,y] \ . \qquad (4.5)$$

where A is an atom and T_1, T_2 are two-variable formulas. Negation normal form transformation (3.25) applied to $\mathcal{FO}^2$ yields a fragment defined by:

$$[\mathcal{FO}^2]^{nnf} \ ::= \ A[x,y] \mid \neg A[x,y] \mid T_1[x,y] \wedge\!\!\!\!\vee T_2[x,y] \mid Qy.T_1[x,y] \ . \qquad (4.6)$$

(recall that $\wedge\!\!\!\!\vee$ denotes conjunction *or* disjunction and Q stands for either the existential or the universal quantifier). A function computing the structural transformation for formulas from $[\mathcal{FO}^2]^{nnf}$, is obtained according to the general procedure (3.9): $[T]^{str}_t := \mathtt{P}_T \wedge [T]^{def}_t$, where:

$$\begin{array}{rrcl}
[T]^{def}_t := & [A[x,y]]^{def}_t & = & \forall xy.(\mathtt{P}_T \rightarrow A[x,y]) \hfill \mid \\
 & [\neg A[x,y]]^{def}_t & = & \forall xy.(\mathtt{P}_T \rightarrow \neg A[x,y]) \hfill \mid \\
 & [T_1[x,y] \wedge\!\!\!\!\vee T_2[x,y]]^{def}_t & = & \forall xy.(\mathtt{P}_T \rightarrow \mathtt{P}_{T_1} \wedge\!\!\!\!\vee \mathtt{P}_{T_2}) \wedge [T_1]^{def}_t \wedge [T_2]^{def}_t \mid \\
 & [Qy.T_1[x,y]]^{def}_t & = & \forall x.(\mathtt{P}_T \rightarrow Qy.\mathtt{P}_{T_1}) \wedge [T_1]^{def}_t \ .
\end{array} \qquad (4.7)$$

Where $\mathtt{P}_T = \mathtt{p}_T[x,y]$ is, as usual, a *definitional predicate* for $T = T[x,y]$. By applying clausification, we obtain clause types (1) – (7) listed in Table 4.4. It is easy to see that clauses of these types satisfy the following properties:

1. *Ground clauses contain at most two different constants;*
2. *Non-functional clauses contain at most two different variables;*
3. *Functional clauses contain at most one variable and (possibly several occurrence of) at most one shallow functional term.*

These properties motivate a clause class (**T**) for two-variable fragment defined in Table 4.5. This clause class is given by three clause schemes. The first clause scheme

Table 4.4 Types of clauses resulted form **CNF** transformation for two-variable formulas

Type of a conjunct	$\dashrightarrow [\cdot]^{sk}, [\cdot]^{cnf} \dashrightarrow$	**Type of a clause**	**(Nr)**
$\exists xy.\mathtt{p}_F[x,y]$	$\dashrightarrow$	$\mathtt{p}_F[\mathtt{c}_1,\mathtt{c}_2]$	(1)
$\forall xy.(\mathtt{p}_F[x,y]\rightarrow a[x,y])$	$\dashrightarrow$	$\neg\mathtt{p}_F[x,y]\vee a[x,y]$	(2)
$\forall xy.(\mathtt{p}_F[x,y]\rightarrow\neg a[x,y])$	$\dashrightarrow$	$\neg\mathtt{p}_F[x,y]\vee\neg a[x,y]$	(3)
$\forall xy.(\mathtt{p}_F[x,y]\rightarrow\mathtt{p}_{F_1}[x,y]\between\mathtt{p}_{F_2}[x,y])$	$\dashrightarrow$	$\neg\mathtt{p}_F[x,y]\vee\mathtt{p}_{F_i}[x,y],\ i=1,2$	(4)
		$\neg\mathtt{p}_F[x,y]\vee\mathtt{p}_{F_1}[x,y]\vee\mathtt{p}_{F_2}[x,y]$	(5)
$\forall xy.[\mathtt{p}_F[x]\rightarrow\forall y.(\mathtt{p}_{F_1}[x,y])]$	$\dashrightarrow$	$\neg\mathtt{p}_F[x]\vee\mathtt{p}_{F_1}[x,y]$	(6)
$\forall xy.[\mathtt{p}_F[x]\rightarrow\exists y.(\mathtt{p}_{F_1}[x,y])]$	$\dashrightarrow$	$\neg\mathtt{p}_F[x]\vee\mathtt{p}_{F_1}[x,\mathtt{f}_{sk}(x)]$	(7)

Table 4.5 A clause class for the two-variable fragment without equality

	Clause scheme	***Description***
	1 $\hat{\alpha}[\mathtt{c}_1,\mathtt{c}_2]$	*a ground clause containing only fixed constants* $\mathtt{c}_1$ *or* $\mathtt{c}_2$*;*
$(\mathbf{T})$:	2 $\hat{\alpha}[x,y]$	*a clause over two variables (without functional symbols);*
	3 $\hat{\alpha}[f(x),x]$	*a clause over one variable and one (fixed) functional term.*
	where $l := p\mid\neg p;\quad \alpha := \vee\{l\}$	

represents ground clauses which contain two fixed constants $\mathtt{c}_1$ and $\mathtt{c}_2$. The second clause scheme represents non-functional clauses that contain at most two variables. The last clause scheme represents clauses that may contain one functional symbol $f(x)$ and at most one variable.

The following analog of Proposition 4.1.3 is easy to prove for the two-variable fragment:

Proposition 4.1.8. *Let $T\in\mathcal{FO}^2$ be a two-variable formula and $\boldsymbol{n}=|T|$. Then* **CNF**-*transformation for T can be computed in polynomial time in $\boldsymbol{n}$ and produces at most $2\boldsymbol{n}$ clauses with at most 3 literals, which belong to* $(\mathbf{T})$.

Saturation of the clause set

Now we have to come up with a saturation strategy which demonstrates that clause class $(\mathbf{T})$ is closed under inference rules of the ordered resolution calculus. The important difference between the clause classes $(\mathbf{T})$ and $(\mathbf{G})$ is that clauses from $(\mathbf{T})$ are no longer guaranteed to have a guard. Therefore, it is not always possible to use the selection function *Sel* to force resolution on literals that contain all variables of a clause. For example, having two clauses of form 1 from $(\mathbf{T})$:

$$1.\ \neg a(x)\vee\underset{\cdots\cdots}{b(x,y)}\qquad 2.\ \neg b(x,x)\vee\underset{\cdots\cdots}{b(x,y)}\tag{4.8}$$

one would like to make the last literals in both these clauses eligible to prevent growth of variables in inferences. For the first clause, we can make the last literal *maximal* by using an ordering $\succ$ that respects *arities* of predicate symbols:

Definition 4.1.9. We say that an ordering $\succ$ on ground literals is *compatible with arities of predicate sybmols* (or short *CAP*) *iff* $l(s_1,..,s_n) \succ k(t_1,..,t_m)$ whenever $max_\succ(s_1,..,s_n) = max_\succ(t_1,..,t_m)$ and $n > m$. ◈

The class of *simple LPO*-orderings described after Definition 4.1.4, can be easily restricted to fulfil the property given in Definition 4.1.9, by requiring the symbols with greater *arity* to be greater in precedence: $p \gg q$ if $ar(p) > ar(q)$. For the purpose of deciding the two-variable fragment, we use the ordered resolution calculus $\mathcal{OR}^\succ$ (without selection) parametrized with a *simple CAP*-ordering $\succ$.

Unfortunately the first literal of the second clause from (4.8) *is always maximal* for *every* admissible ordering $\succ$. Indeed, in instance $\neg b(c,c) \vee b(c,c)$ of this clause, the first literal is maximal because of condition (R1) of admissible orderings (see Definition 3.5.1 on p. 79).

Several techniques have been proposed in literature to avoid this problem. This problem can be avoided by using *non-liftable orders*, similar to (4.1) considered in subsection 4.1.1, and employing resolution strategies that decide classes $\mathcal{S}^+$ and $\mathcal{E}^+$ [Tammet, 1990; Fermüller et al., 1993; de Nivelle, 1995, 2000*b*,*a*].

Another possibility is to use a refinement of the ordered resolution called the *lock resolution* [Boyer, 1971] (see also [Bachmair & Ganzinger, 2001]). According to this refinement, one should supply additional *lock indexes* for literals in clauses (which are in our case the number of different variables in a literal), and use an ordering defined on the *indexed literals*. This strategy for the two-variable fragment has been described in [de Nivelle & Pratt-Hartmann, 2001]. The *lock resolution* is in somewhat similar to non-liftable orders, although completeness for this refinement is justified using *renaming techniqes* by a reduction to the classical ordered resolution calculus [see Bachmair & Ganzinger, 2001].

Renaming techniques can be applied to clauses for the two-variable fragment *directly*, which gives a third possibility of treating the problem above. For the particular example (4.8), one can introduce a "name" $p_{\neg b(\cdot,\cdot)}(x)$ for the first literal of the second clause and *split* this clause into two clauses, in which maximal literals now contain all variables of the clause:

$$2.1 \quad p_{\neg b(\cdot,\cdot)}(x) \vee \underline{\boldsymbol{b(x,y)}}^\star \qquad 2.2 \quad \neg p_{\neg b(\cdot,\cdot)}(x) \vee \underline{\boldsymbol{\neg b(x,x)}}^\star \tag{4.9}$$

Renaming strategies of a similar sort have been employed in [Hustadt, 1999; Hustadt & Schmidt, 1999] for deciding clause classes related to the two-variable fragment.

We may notice that literal renaming is reminiscent of rule Splitting through New Predicate Symbol (3.23) discussed in 3.5.6. We can achieve the same effect as above by using an instance of this rule, which we call the Literal Projection rule: see Figure 4.1. According to this rule, a clause is split into two clauses through a new predicate $p_{l[\cdot]}(x)$ that is introduced for a literal $l[x]$ containing duplicate occurrences of x. If our ordering $\succ$ is *simple* and *compatible with arities of predicate symbols*, and if the premise of this rule contains some other non-unary or functional literals with x (which are then greater than $p_{l[\cdot]}(x)$), the premise of this rule becomes redundant after the inference is made, so it is a *simplification rule extending the signature*. Note that new predicate symbols cannot be introduced for literals that already contain those. So the extension of a signature by this rule is always finite. Since Literal Projection is an optional rule, i.e., it does not affect refutational completeness of the calculus, we can apply this rule "on demand", i.e., when it is needed in order to avoid potential problems.

Figure 4.1 The Literal Projection rule

Literal Projection

$$\mathrm{LP}: \frac{[\![C \vee \boldsymbol{l}[\boldsymbol{x}]^{\sharp}]\!]}{\begin{array}{r} C \vee p_{l[\cdot]}(x) \\ \neg p_{l[\cdot]}(x) \vee l[x] \end{array}}$$

$$\left[\begin{array}{l}\textit{where (i) } l[x] \textit{ is a non-unary literal and (ii) } C \textit{ has a} \\ \textit{non-unary literal or a functional term, containing } x.\end{array}\right]$$

Using the Literal Projection rule formulated in Figure 4.1 has an advantage over non-liftable orders and lock resolution, in that we stay within our general framework of saturation-based theorem proving, and, in particular, redundancy elimination strategies can be applied as usual. Moreover, this rule is not intended to work only as a preprocessor of the input clauses, like renaming in [Hustadt, 1999; Hustadt & Schmidt, 1999], but can interleave with the saturation process. This is not required for the two-variable fragment, where it can be shown that Literal Projection might be applied only to the input clauses, but we will consider fragments, in which this rule should be applied dynamically to newly generated clauses.

The case analysis for possible inferences between clauses from (**T**) is summarised in Table 4.6. According to this strategy, for ground clauses of type 1, we first exhaustively apply the Splitting rule (case 1.0), until clauses with only one literal are obtained, which are then subject to further resolution inferences (case 1.1).

For convenience, we have separated clauses of type 2 that contain at most one variable into case 4 (case 2.0). For the remaining ones the following cases are possible: Case 2.1 is considered when the eligible literal of a clause contains *both* variables x and y. In this case, the Ordered Resolution and the Ordered Factoring rules can be applied. In case 2.2, the eligible literal contains one variable, but there

Table 4.6 Possible inferences between clauses for the two-variable fragment

1	$\hat{\alpha}[\mathsf{c}_1,\mathsf{c}_2]$	
$\bot$	$\Box$	
1.0	$[\![\, \hat{\alpha}[\mathsf{c}_1,\mathsf{c}_2] \vee l[\mathsf{c}_1,\mathsf{c}_2] \,]\!]$	$:\mathtt{SP}$
1.1	$\underline{\boldsymbol{l}[\mathsf{c}_1,\mathsf{c}_2]}^{\star}$	$:\mathtt{OR}$
	$\mathtt{OR}[1.1;1.1]\colon \Box \colon \bot$	
2	$\hat{\alpha}[x,y]$	
2.0	$\hat{\alpha}[x] \quad \Rightarrow \; 4$	
2.1	$\hat{\alpha}[x,y] \vee \hat{\boldsymbol{l}}[!\boldsymbol{x},!\boldsymbol{y}]$	
2.1.1	$\hat{\alpha}[x,y] \vee \underline{\boldsymbol{l}[!\boldsymbol{x},!\boldsymbol{y}]}^{\star}$	$:\mathtt{OR}$
2.1.3	$\hat{\alpha}[x,y] \vee \underline{p[x,y]} \vee \underline{\boldsymbol{p}[\boldsymbol{x},\boldsymbol{y}]}$	$:\mathtt{OF}$
	$\mathtt{OR}[2.1.1;1.1] \;\colon \hat{\alpha}[\hat{c}] \quad :1$	
	$\mathtt{OR}[2.1.1;2.1.1]\colon \hat{\alpha}[x,y]\colon 2$	
	$\mathtt{OF}[2.1.3]\colon \hat{\alpha}[x,y] \vee p[x,y]\colon 2$	
2.2	$[\![\, \hat{\alpha}[x,y] \vee \boldsymbol{l}[!\boldsymbol{x}]^{\sharp} \,]\!]$	$:\mathtt{LP}$
	$\mathtt{LP}[2.2]\colon \hat{\alpha}[x,y] \vee p_{l[\cdot]}(x)\colon 2$ $:\neg p_{l[\cdot]}(x) \vee l[!x] \;:4$	
2.3	$[\![\, \hat{\alpha}[x] \vee \hat{\alpha}[y] \,]\!]$	$:\mathtt{SP}$
	$\mathtt{SP}[2.3]\colon \hat{\alpha}[x] : 4 \parallel \hat{\alpha}[y] : 4$	
3	$\hat{\alpha}[f(x),x]$	
3.1	$\hat{\alpha}[f(x),x] \vee \boldsymbol{l}[!\boldsymbol{f}(\boldsymbol{x}),\boldsymbol{x}]$	
3.1.1	$\hat{\alpha}[f(x),x] \vee \underline{\boldsymbol{l}[!\boldsymbol{f}(\boldsymbol{x}),\boldsymbol{x}]}^{\star}$	$:\mathtt{OR}$
	$\mathtt{OR}[2.1.1;3.1.1]\colon \hat{\alpha}[f(x),x] \vee \hat{\alpha}[f(x),x]\colon 3$	
	$\mathtt{OR}[3.1.1;3.1.1]\colon \hat{\alpha}[f(x),x] \quad :3$	
4	$\hat{\alpha}[x]$	
4.1	$\hat{\alpha}[x] \vee \boldsymbol{l}[!\boldsymbol{x}]$	
4.1.1	$\hat{\alpha}[x] \vee \underline{\boldsymbol{l}[!\boldsymbol{x}]}^{\star}$	$:\mathtt{OR}$
	$\mathtt{OR}[4.1.1;1.1] \;\colon \hat{\alpha}[\mathsf{c}_1,\mathsf{c}_2] \quad :1$	
	$\mathtt{OR}[4.1.1;2.1.1]\colon \hat{\alpha}[x] \vee \hat{\alpha}[x] \quad :4$	
	$\mathtt{OR}[4.1.1;3.1.1]\colon \hat{\alpha}[f(x)] \vee \hat{\alpha}[f(x),x]\colon 3$	
	$\mathtt{OR}[4.1.1;4.1.1]\colon \hat{\alpha}[x] \quad :4$	
4.2	$\hat{\alpha} \quad \Rightarrow \; 1$	

are literals that contain both variables. This is only possible if the eligible literal contains duplicate occurrences of the variable, in which case we reduce the clause by applying the Literal Projection rule. In the remaining case 2.3, every literal of a clause contains at most one variable, but the clause contains two variables. In this case the clause can be split into variable disjoint parts using the Splitting rule.

Clauses of type 3 are functional clauses with one variable. The case analysis of inferences for these clauses is straightforward.

Clauses of type 4 contain at most one variable and no functional term. Hence either the maximal literal of such a clause contains its variable (case 4.1), or otherwise, there is no variable in this clause, and so the clause is ground (case 4.2).

The case analysis summarised in Table 4.6 proves the following lemma:

Lemma 4.1.10 (Closure of (**T**))**.** *There is a strategy based on ordered resolution that given a set N of clauses of form* (**T**) *computes its saturation N', which consists of clauses of form* (**T**) *constructed over the signature of N extended possibly with unary predicate symbols of form p_L, where $L = a[x]$, or $L = \neg a[x]$ for some predicate symbol a that occurs in N.*

By carrying out the complexity calculations (see Appendix C.1.2) similar to those for the guarded fragment, we obtain the following bound on the non-deterministic running time t of our procedure:

$$\boxed{\boldsymbol{t} \;\leq\; 2^{O(\boldsymbol{n})}}$$

where $n := |T|$ is the size of the input formula $T \in \mathcal{FO}^2$. As a consequence, the following theorem holds:

Theorem 4.1.11. *There is a resolution-based decision procedure for the two-variable fragment without equality $\mathcal{FO}^2$ of NEXPTIME (optimal) complexity.*

Note 4.1.12. *As has been pointed out in subsection 3.3.2, Grädel, Kolaitis & Vardi [1997] demonstrated that every two-variable formula with equality is satisfiable in a model of size $2^{O(\boldsymbol{n})}$ (the same bound on the model size for the case without equality follows from a simpler estimation for the Gödel class given in [Börger et al., 1997]). This means that both procedures—those based on enumeration of finite models and based on saturation—have essentially the same* worst-case *complexity. However, refinements of saturation-based procedures make our procedure more efficient for the average case.*

Although there are lots of works on resolution decision procedures that subsume our decidability *result (see the beginning of this section), to the best of our knowledge, complexity of the resulted decision procedures have not been estimated.* ◇

4.1.3 Deciding the Monadic Fragments without Equality

In this section we demonstrate how our method can be extended for obtaining a resolution-based decision procedure for the *monadic fragment without equality* $\mathcal{M}$.

A first resolution-based strategy that decides the monadic fragment $\mathcal{M}$ without equality, has been proposed by Joyner Jr. [1976]. Later in [Bachmair et al., 1993*b*] this procedure has been extended to the case without equality by employing the *superposition calculus* with specially designed simplification rules. Both papers define a special clause class for monadic formulas (called *MON* in the first paper and *flat clauses* in the second one) and describe a saturation procedure which preserves the clauses of this form. We demonstrate a relationship between monadic formulas and *flat clauses* defined in [Bachmair et al., 1993*b*] by an example.

Definition 4.1.13. A clause C is *flat* *iff*
(***i***) all atoms from C are unary or equational (in the case with equality), and
(***ii***) there is a list $\overline{v} = v_1,..,v_n$ where v_i with $1 \leq i \leq n$ is either a variable or a constant, such that every functional subterm of C has form $f(v_1,..,v_k)$, where $0 \leq k \leq n$. ◇

For example, the monadic formula (without equality)

$$\exists \mathtt{u}.\forall \mathtt{x}.([\mathtt{a}(\mathtt{u}) \wedge \mathtt{b}(\mathtt{x})] \vee \exists \mathtt{v}.[\mathtt{p}(\mathtt{v}) \wedge \forall \mathtt{y}.(\mathtt{q}(\mathtt{y}) \wedge \exists \mathtt{z}.\mathtt{r}(\mathtt{z}))]) \qquad (4.10)$$

is first put into a *prenex normal form*:

$$\exists \mathtt{u}.\forall \mathtt{x}.\exists \mathtt{v}.\forall \mathtt{y}.\exists \mathtt{z}.([\mathtt{a}(\mathtt{u}) \wedge \mathtt{b}(\mathtt{x})] \vee [\mathtt{p}(\mathtt{v}) \wedge \mathtt{q}(\mathtt{y}) \wedge \mathtt{r}(\mathtt{z})])$$

and then is *skolemaized* in the outermost way ($\mathtt{u} \rightarrow \mathtt{c}$, $\mathtt{v} \rightarrow \mathtt{f}_1(\mathtt{x})$, $\mathtt{z} \rightarrow \mathtt{f}_2(\mathtt{x}, \mathtt{y})$) into the formula

$$[\mathtt{a}(\mathtt{c}) \wedge \mathtt{b}(\mathtt{x})] \vee [\mathtt{p}(\mathtt{f}_1(\mathtt{x})) \wedge \mathtt{q}(\mathtt{y}) \wedge \mathtt{r}(\mathtt{f}_2(\mathtt{x}, \mathtt{y}))]$$

which yields the following *flat clauses*:

$$\begin{array}{lll} \mathtt{a}(\mathtt{c}) \vee \mathtt{p}(\mathtt{f}_1(\mathtt{x})), & \mathtt{a}(\mathtt{c}) \vee \mathtt{q}(\mathtt{y}), & \mathtt{a}(\mathtt{c}) \vee \mathtt{r}(\mathtt{f}_2(\mathtt{x}, \mathtt{y})), \\ \mathtt{b}(\mathtt{x}) \vee \mathtt{p}(\mathtt{f}_1(\mathtt{x})), & \mathtt{b}(\mathtt{x}) \vee \mathtt{q}(\mathtt{y}), & \mathtt{b}(\mathtt{x}) \vee \mathtt{r}(\mathtt{f}_2(\mathtt{x}, \mathtt{y})). \end{array}$$

A disadvantage of this transformation to *flat clauses* which can be immediately observed, is that it may result in *exponentially* many clauses in the size of the input monadic formula (it is easy to construct a simple example that exhibits this behaviour similar to (3.27) given on p. 93). We describe an alternative procedure that employs a *structural transformation* and avoids this exponential blowup. Moreover, we show how to decide the *full monadic class* and prove that our decision procedure has the best known complexity.

A clause class for the monadic fragment

As usual, we start from a recursive definition for the *full monadic class* given in (3.13) on p. 72:

$$\mathcal{M}_f \ ::= \ A[x] \mid M_1[x] \cdot \{x/f(x)\} \mid \neg M_1 \mid M_1 \wedge M_2 \mid \forall x.M_1 \ . \tag{4.11}$$

where $A[x]$ is a unary atom and M_1, M_2 are monadic formulas. In fact, it does not really matter whether atoms $A[x]$ are unary, the main point is that they should contain *at most one variable*. By applying the usual **NNF**-transformation to $\mathcal{M}_f$ we obtain the fragment:

$$[\mathcal{M}_f]^{nnf} \ ::= (\neg)A[x] \mid M_1[x] \cdot \{x/f(x)\} \mid M_1 \mathbin{\wedge\mkern-13mu\vee} M_2 \mid Qx.M_1 \ . \tag{4.12}$$

A function computing the structural transformation is defined recursively over (4.12) as usual, according to Figure 3.9 on p.94: $[M]^{str}_m := \mathtt{P}_M \wedge [M]^{def}_m$, where:

$$\begin{array}{rcll} [M]^{def}_t := [(\neg)A[x]]^{def}_t & = & \forall x.(\mathtt{P}_M \rightarrow (\neg)A[x]) & | \\ [M_1[x] \cdot \{x/f(x)\}]^{def}_t & = & \forall x.(\mathtt{P}_M \rightarrow \mathtt{P}_{M_1}[x/f(x)]) \wedge [M_1]^{def}_t & | \\ [M_1 \mathbin{\wedge\mkern-13mu\vee} M_2]^{def}_t & = & \forall \overline{x}.(\mathtt{P}_M \rightarrow \mathtt{P}_{M_1} \mathbin{\wedge\mkern-13mu\vee} \mathtt{P}_{M_1}) \wedge [M_1]^{def}_t \wedge [M_1]^{def}_t & | \\ [Qy.M_1]^{def}_t & = & \forall \overline{x}.(\mathtt{P}_M \rightarrow Qy.\mathtt{P}_{M_1}) \wedge [M_1]^{def}_t \ . & \end{array} \tag{4.13}$$

However, here we introduce the definitional predicates $\mathtt{P}_{M'}$ for sumbformulas M' of M in a slightly different way. W.l.o.g. we may assume that M is a sentence (i.e., it contains no free variables). Definitional predicates for subformulas of M that match first two cases of (4.13) are introduced as usual over all free variables of the defined formula. However, for the remaining two cases, we introduce definitional predicates in the *outermost way*, i.e., variables of $\mathtt{P}_{M'}$ are all variables in *scope of which* subformula M' occurs in M, and they are arranged according to the *first appearance* of variables in M. For example, definitional predicates for formula (4.10) are introduced as indicated below:

$$\exists \mathtt{u}.\forall \mathtt{x}.(\underbrace{[\overbrace{\mathtt{a(u)}}^{\mathtt{P_a(u)}} \wedge \overbrace{\mathtt{b(x)}}^{\mathtt{P_b(x)}}]}_{\mathtt{P_1(u,x)}} \vee \overbrace{\exists \mathtt{v}.[\overbrace{\mathtt{p(v)}}^{\mathtt{P_p(v)}} \wedge \underbrace{\forall \mathtt{y}.(\overbrace{\mathtt{q(y)}}^{\mathtt{P_q}(y)} \wedge \overbrace{\exists \mathtt{z.r(z)}}^{\mathtt{P_3(u,x,v,y)}})}_{\mathtt{P_2(u,x,v)}}]}^{\mathtt{P_5(u,x)}})$$

It is easy to justify such structural transformation in the same way as for the usual structural transformation given in (3.9). Taking into account the form of the definitional predicates, from (4.13) we obtain clause types (1) – (7) listed in Table 4.7. Note that variables of clauses occur in a *fixed order* in every literal. This will be the essential property used by our decision procedure. We define a clause class (**M**) for monadic fragment in Table 4.8. Note how we have separated usage of definitional predicates form monadic formulas (clause parameter α_m) and unary definitional predicate that originate from clause types (2) and (3) in Table 4.7.

Table 4.7 Types of clauses resulted form **CNF** transformation for monadic formulas

Type of a conjunct	$\dashrightarrow [\cdot]^{sk}, [\cdot]^{cnf} \dashrightarrow$	**Type of a clause**	(**Nr**)
$\mathtt{p}_M$	$\dashrightarrow$	$\mathtt{p}_M$	(1)
$\forall x.(\mathtt{p}_M(x) \to (\neg)a[x])$	$\dashrightarrow$	$\neg\mathtt{p}_M(x) \vee (\neg)a[x]$	(2)
$\forall x.[\mathtt{p}_M(x) \to \mathtt{p}_{M_1}(f(x))]$	$\dashrightarrow$	$\neg\mathtt{p}_M(x) \vee \mathtt{p}_{M_1}(f(x))$	(3)
$\forall \overline{x}.(\mathtt{p}_M(\overline{x}) \to \mathtt{p}_{M_1}(\overline{x}) \mathbin{\wedge\kern-0.75em\vee} \mathtt{p}_{M_2}(\overline{x}))$	$\dashrightarrow$	$\neg\mathtt{p}_M(\overline{x}) \vee \mathtt{p}_{M_i}(\overline{x}),\ i = 1, 2$	(4)
		$\neg\mathtt{p}_M(\overline{x}) \vee \mathtt{p}_{M_1}(\overline{x}) \vee \mathtt{p}_{M_2}(\overline{x})$	(5)
$\forall \overline{x}.[\mathtt{p}_M(\overline{x}) \to \forall y.\mathtt{p}_{M_1}(\overline{x}, y)]$	$\dashrightarrow$	$\neg\mathtt{p}_M(\overline{x}) \vee \mathtt{p}_{M_1}(\overline{x}, y)$	(6)
$\forall \overline{x}.[\mathtt{p}_M(\overline{x}) \to \exists y.\mathtt{p}_{M_1}(\overline{x}, y)]$	$\dashrightarrow$	$\neg\mathtt{p}_M(\overline{x}) \vee \mathtt{p}_{M_1}(\overline{x}, \mathtt{f}_{sk}(\overline{x}))$	(7)

Proposition 4.1.14. *Let $M \in \mathcal{M}$ be a monadic formula and $\boldsymbol{n} = |M|$. Then* **CNF**-*transformation for M can be computed in polynomial time in $\boldsymbol{n}$ and produces at most $2\boldsymbol{n}$ clauses with at most 3 literals, which belong to* (**M**).

Table 4.8 A clause class for the monadic fragment without equality

	Clause scheme	*Description*
(**M**):	1 $\hat{\alpha}[f(\overline{x}),\overline{x}]$, $\lvert\overline{x}\rvert \leq 1$	*a clause with at most one variable and at most one functional term containing all variables in a fixed order*
	2 $\hat{\alpha}_m(\overline{x},y) \vee \hat{\alpha}_m(\overline{x}) \vee \hat{\alpha}^1_m[\overline{x},y]$	*a clause containing only definitional predicates for monadic formulas with arguments* $(\overline{x})$ *and* $(\overline{x},y)$, *and unary definitional predicates whose argument is either from* $\overline{x}$ *or is* y
	3 $\hat{\alpha}_m(\overline{x},f(\overline{x}))\vee\hat{\alpha}_m(\overline{x})\vee\hat{\alpha}^1_m[\overline{x},f(\overline{x})]$	*the same as in the previous case, but instead of variable* y *there is a functional term* $f(\overline{x})$
	4 $\hat{\alpha}_m(\overline{x}) \vee \hat{\alpha}[f(\overline{x})] \vee \hat{\alpha}^1_m[\overline{x}]$	*a clause that contains definitional predicates with arguments* $(\overline{x})$, *predicates containing only fixed functional term* $f(\overline{x})$, *and unary definitional predicates with an argument from* $\overline{x}$
	where $l := p \mid \neg p$; $\alpha := \vee\{l\}$; $l_m := p_m \mid \neg p_m$; $\alpha_m := \vee\{l_m\}$	

Saturation of the clause set

For deciding the monadic class we employ the same resolution strategy as has been applied for clauses of the two-variable fragment, except that we don't need the Literal Projection rule anymore. However, we still employ a *simple CAP* ordering $\succ$ in $\mathcal{OR}^{\succ}$.

All possible resolution inferences between clauses from (**T**) are summarised in Table 4.9. These inferences are quite straightforward and no "tricks" (like additional simplification rules) have been used, except for the Splitting rule in case 2.3.

Table 4.9 Possible inferences between clauses for the two-variable fragment

1 $\hat{\alpha}[f(\overline{x}),\overline{x}]$, $\lvert\overline{x}\rvert \leq 1$		3 $\hat{\alpha}_m(\overline{x},f(\overline{x}))\vee\hat{\alpha}_m(\overline{x})\vee\hat{\alpha}^1_m[\overline{x},f(\overline{x})]$	
1.1 $\hat{\alpha}[f(\overline{x}),\overline{x}]\vee\underline{\boldsymbol{\hat{l}}[!\boldsymbol{f}(\overline{\boldsymbol{x}}),\overline{\boldsymbol{x}}]}$	:OR	3.1 $\hat{\alpha}_m(\overline{x},f(\overline{x}))\vee\hat{\alpha}_m(\overline{x})\vee\hat{\alpha}^1_m[\overline{x},f(\overline{x})]\vee\underline{\boldsymbol{\hat{l}_m}(\overline{\boldsymbol{x}},\boldsymbol{f}(\overline{\boldsymbol{x}}))}$	:OR
1.2 $\hat{\alpha}[\overline{x}] \vee \underline{\boldsymbol{\hat{\alpha}}[!\overline{\boldsymbol{x}}]}$	:OR	3.2 $\hat{\alpha}_m(\overline{x}) \vee \hat{\alpha}^1_m[\overline{x},f(\overline{x})]$	:4
OR[1.∗; 1.∗]: 1		OR[1.∗, 3.1] : 1; OR[2.∗, 3.1] : 3; OR[3.1, 3.1] : 3:	
2 $\hat{\alpha}_m(\overline{x},y)\vee\hat{\alpha}_m(\overline{x})\vee\hat{\alpha}^1_m[\overline{x},y]$		4 $\hat{\alpha}_m(\overline{x}) \vee \hat{\alpha}[f(\overline{x})]\vee\hat{\alpha}^1_m[\overline{x}]$	
2.1 $\hat{\alpha}_m(\overline{x},y)\vee\hat{\alpha}_m(\overline{x})\vee\hat{\alpha}^1_m[\overline{x},y]\vee\underline{\boldsymbol{\hat{l}_m}(\overline{\boldsymbol{x}},\boldsymbol{y})}$	:OR	4.1 $\hat{\alpha}_m(\overline{x}) \vee \hat{\alpha}[f(\overline{x})]\vee\hat{\alpha}^1_m[\overline{x}]\vee\underline{\boldsymbol{\hat{l}}[!\boldsymbol{f}(\overline{\boldsymbol{x}})]}$	:OR
2.2 $\hat{\alpha}_m(\overline{x}) \vee \hat{\alpha}^1_m[\overline{x}] \vee \underline{\boldsymbol{\hat{l}_m}(\overline{\boldsymbol{x}})}$	:OR	4.2 $\hat{\alpha}_m(\overline{x}) \vee \hat{\alpha}^1_m[\overline{x}]$	:2
2.3 $[\![\, \hat{\alpha}_m(\overline{x}) \vee \hat{\alpha}^1_m[\overline{x}] \vee \hat{\alpha}^1_m[y] \,]\!]$	:SP	OR[1.1, 4.1] : 1; OR[1.2, 4.1] : 4;	
OR[1.∗, 2.∗] : 1; OR[2.∗, 2.∗] : 2		OR[2.1, 4.1] : 4; OR[2.2, 4.1] : 4;	
SP[2.3]: $\hat{\alpha}_m[\overline{x}] \vee \hat{\alpha}^1_m[\overline{x}]$: 2 ‖ $\hat{\alpha}^1_m[y]$: 1		OR[3.1, 4.1] : 1; OR[4.1, 4.1] : 4;	

For clauses described by scheme 1 there are two cases possible. Either there exists a functional literal in this clause, and hence such a literal must be maximal (case 1.1), or, otherwise all literals are *shallow*, and hence the maximal literal must contain all variables of the clause, since the number of variables is at most 1 (case 1.2).

For the clauses that correspond to scheme 2, the following cases are possible. Either there exists a literal that contains all variables, and hence such a literal must be maximal, since it is of the maximal arity (case 2.1 and case 2.2), or, otherwise, the clause can be split into variable disjoint parts (case 2.3).

A clause that correspond to scheme 3, either contains a literal of form $l_m(\overline{x}, f(\overline{x}))$, and hence such a literal must be maximal (case 3.1), or, otherwise, the clause is captured by scheme 4.

For clauses that are described by scheme 4, there is either a functional literal, and hence such a literal must be maximal, since the functional term contains all variables of a clause (case 4.1), or, otherwise, there are no functional terms in this clause, and so the case is analogous to 2.2.

Note that no Ordered Factoring rule can be applied to clauses from (**M**): otherwise a clause can be always simplified by Elimination of Duplicate Literals, which we assume to be applied eagerly. The case analysis summarised in Table 4.9, proves the following lemma:

Lemma 4.1.15 (Closure of (**M**))**.** *There is a strategy based on ordered resolution that given a set N of clauses of form* (**M**)*, computes its saturation N', which consists of clauses of form* (**M**) *constructed over the signature of N.*

By analysing the complexity of the procedure given in Table 4.9, we obtain that the non-deterministic running time of our procedure is bounded by: $\boxed{\boldsymbol{t} \leq 2^{O(\boldsymbol{n})}}$, where $n := |M|$ is the size of the input formula $M \in \mathcal{M}_f$ (see Appendix C.1.2 for the details). Hence the following result is proven:

Theorem 4.1.16. *There is a resolution-based decision procedure for the full monadic class $\mathcal{M}_f$ of NEXPTIME (optimal) complexity.*

4.2 Combinations of Decidable Fragments

Combination of algorithms, and in particular decision procedures, is nowadays one of the central topic in automated deduction. For decidable fragments of first-order logic, the problem can be formulated as follows: *find a method of forming a new decidable fragment from known ones so that their respective decision procedures can be reused for the combined fragment.* The purpose of combining decidable fragments is quite obvious: to identify more expressible subclasses of first-order formulas and design decision procedures for them in a modular way.

There is not much known about combination of decidable fragments of first-order logic. Some (mostly undecidability) results about Boolean combinations of *prefix-vocabulary classes* are mentioned in [Börger et al., 1997]. One of the known decidable classes obtained in this way is the *Scott class* consisting of Boolean combinations of formulas with quantifier prefixes ∀∀ and ∀∃ [see Grädel, Kolaitis & Vardi, 1997]. We have implicitly considered this class in connection with the *two-variable fragment* (see subsection 3.3.2).

In this thesis we introduce a different type of combination of first-order fragments, motivated by what one usually observes in logical formalisms like *modal logics*, *description logics*, *dynamic logics*, etc.. These logics are usually defined *recursively* using different types of *constructors* (see section 3.2). Extensions and combinations of these logics are formed by *combining their constructors*—this is especially emphasised for *description logics*: see Table 3.2 on p.67.

Recursive definitions for the two-variable, guarded and monadic fragments that have been considered in this chapter, can be seen as collections of "safe" constructors for first-order formulas. Decidability of these fragments is achieved by quite orthogonal means: (*i*) in the guarded fragment, the form of quantification is restricted; (*ii*) in the two-variable fragment, only formulas containing two free variables are constructed, whereas (*iii*) in the monadic fragment, only unary predicate symbols are allowed. At this point the reader should agree that it is reasonable to ask ourselves, *what if we join these constructors in one recursive definition*?

For example we may consider a fragment that is defined by:

$$\mathcal{GF}|\mathcal{FO}^2 \ ::= \ A \mid \neg F_1 \mid F_1 \wedge F_2 \mid \forall \overline{y}.[G \rightarrow F_1] \mid \forall y.F_1[x,y] \ . \tag{4.14}$$

where we took a recursive definition (3.17) for the *guarded fragment* augmented with a constructor from the recursive definition for the two-variable fragment (3.15), which is not subsumed by other constructors. Obviously, we obtain a fragment that is more expressive than both the guarded-fragment and the two-variable fragment. Note an important difference between $\mathcal{GF}|\mathcal{FO}^2$ and fragment $\mathcal{GF}^2$: the last is an intersection of the two-variable and the guarded fragments, whereas the first is a fragment that contains their union.

We will call such a combination of fragments, a *structural combination*, meaning that the structures of underlying fragments have been combined. It is rather natural to consider structural combinations of fragment in order to study decidability of expressive logical formalisms through the first-order logic.

However, it is not very clear how to define a structural combination with the monadic fragment, in particular, how to apply the restrictions on the arity of predicate symbols used in this fragment: obviously, if we restrict the arity of *all* predicate symbols, we simply obtain the *monadic fragment*. In order to understand the situation better, and to come up with a formal definition for the *structural combination* of fragments, consider the formula:

$$\underbrace{\forall \mathtt{x}.[\mathtt{Nat(x)} \rightarrow \mathtt{Nat(s(x))}]}_{\in \mathcal{M}_f} \wedge \underbrace{\forall \mathtt{xy}.[\mathtt{Nat(x)} \wedge \mathtt{Nat(y)} \rightarrow \overbrace{\exists \mathtt{z}.(\mathtt{Sum(x,y,z)} \wedge \mathtt{Nat(z)})}^{\mathtt{Summable(x,y)} \in \mathcal{GF}}]}_{\in \mathcal{GF}|\mathcal{FO}^2} \tag{4.15}$$

This formula expresses some properties of natural numbers: existing of a successor element and a sum of natural numbers. It is easy to see that the first conjunct in this formula is a formula from the *full monadic fragment*. In the second conjunct we find a subformula $\exists \mathtt{z}.(\mathtt{Sum}(\mathtt{x},\mathtt{y},\mathtt{z}) \wedge \mathtt{Nat}(\mathtt{z}))$ which is from the *guarded fragment*. If we now treat this subformula as a new binary atom $\mathtt{Summable}(\mathtt{x},\mathtt{y})$ with two variables, we can easily see that the second conjunct forms a two-variable formula in this notation. Hence, the second conjunct belongs to the structural combination of the *guarded fragment* and the *two-variable fragment* $\mathcal{GF}|\mathcal{FO}^2$ defined in (4.14). So the structural combination of fragments can be described operationally as follows: (***i***) define a formula in one fragment, (***ii***) pass this formula like a new atom to the other fragment, and (***iii***) repeat again from step (*i*).

Formally, given recursive definitions for first-order fragments:

$$\begin{array}{lcl} \mathcal{F}_1 & ::= & B^1_1[A_1,..] \mid \cdots \mid B^1_{b_1}[A_1,..] \mid R^1_1[F^1_1,..] \mid \cdots \mid R^1_{r_1}[F^1_1,..] \ . \\ & \cdots\cdots\cdots & \\ \mathcal{F}_k & ::= & B^k_1[A_1,..] \mid \cdots \mid B^k_{b_k}[A_1,..] \mid R^k_1[F^k_1,..] \mid \cdots \mid R^k_{r_k}[F^k_1,..] \ . \end{array} \tag{4.16}$$

where for every i with $1 \leq i \leq k$, elements $B^i_j[A_1,..]$ with $1 \leq j \leq b_i$, correspond to *base constructors* defined for atoms, and elements $R^i_j[F^i_1,..]$ with $1 \leq j \leq r_i$, correspond to *recursive constructors* for already defined formulas $F^j_1 \in \mathcal{F}_j$, etc.. Now a *structural combination* $\mathcal{F}_1|\cdots|\mathcal{F}_k$ *of fragments* $\mathcal{F}_1, \ldots, \mathcal{F}_k$ is defined by joining the definitions from (4.16) as follows:

$$\begin{array}{lcl} \mathcal{F}_1|\cdots|\mathcal{F}_k & ::= & A \mid F^1_1 \mid \cdots \mid F^k_1 \ . \\ \mathcal{F}'_1 & ::= & B^1_1[P_1,..] \mid \cdots \mid B^1_{b_1}[P_1,..] \mid R^1_1[F^1_1,..] \mid \cdots \mid R^1_{r_1}[F^1_1,..] \ . \\ \cdots\cdots\cdots & & \\ \mathcal{F}'_k & ::= & B^k_1[P_1,..] \mid \cdots \mid B^k_{b_k}[P_1,..] \mid R^k_1[F^k_1,..] \mid \cdots \mid R^k_{r_k}[F^k_1,..] \ . \end{array} \tag{4.17}$$

where A is any atom, $F^i_1,.. \in \mathcal{F}'_i$, $1 \leq i \leq k$, and $P_1,.. \in \mathcal{F}_1|\cdots|\mathcal{F}_k$.

For example, a recursive combination of the guarded fragment, two-variable fragments and the full monadic class is defined using (3.17), (3.15) and (3.11) according to (4.17) as follows:

$$\begin{aligned} \mathcal{GF}|\mathcal{FO}^2|\mathcal{M}_f \ &::= \quad A \mid F_1 \mid T_1 \mid M_1 \ . \\ \mathcal{GF}' \ &::= \quad P \mid \neg F_1 \mid F_1 \wedge F_2 \mid \forall \overline{y}.[G \rightarrow F_1] \ . \\ \mathcal{FO}^{2'} \ &::= \ P[x,y] \mid \neg T_1 \mid T_1[x,y] \wedge T_2[x,y] \mid \forall y.T_1[x,y] \ . \\ \mathcal{M}_f{}' \ &::= \quad P[x] \mid M_1[x] \cdot \{x/f(x)\} \mid \neg M_1 \mid M_1 \wedge M_2 \mid \forall x.M_1 \ . \end{aligned} \tag{4.18}$$

where $F_1, F_2 \in \mathcal{GF}'$, $T_1, T_2 \in \mathcal{FO}^{2'}$, $M_1, M_2 \in \mathcal{M}_f{}'$ and $P \in \mathcal{GF}|\mathcal{FO}^2|\mathcal{M}_f$. Note that the atom G in the guard position has *not* been replaced by P, since this constructor does not correspond to the base case. This indicates an important point, namely that a structural combination is defined for *definitions* of fragments rather than their sets of formulas. Since same fragments can be possibly represented by different recursive definitions, their combinations may vary considerably.

Since the combined fragments inherit certain features of their components, it is reasonable to expect that their good computational properties are also preserved, in particular decidability and complexity. As will be demonstrated below, this is indeed the case for many combinations of the fragments that we have considered. Moreover, the resolution-based decision procedures for these fragments described in the previous sections, can be *reused* for their combinations.

4.2.1 Deciding the Combination of Guarded and Two-Variable Fragments

In this section we demonstrate how to obtain a resolution decision procedure for the structural combination $\mathcal{GF}|\mathcal{FO}^2$ of the *guarded* and *two-variable* fragments from those for their components.

We start from a recursive definition for $\mathcal{GF}|\mathcal{FO}^2$, which is obtained according to the general scheme (4.16):

$$\begin{aligned} \mathcal{GF}|\mathcal{FO}^2 \ &::= \quad A \mid F_1 \mid T_1 \ . \\ \mathcal{GF}' \ &::= \quad P \mid \neg F_1 \mid F_1 \wedge F_2 \mid \forall \overline{y}.[G \rightarrow F_1] \ . \\ \mathcal{FO}^{2'} \ &::= \ P[x,y] \mid \neg T_1 \mid T_1[x,y] \wedge T_2[x,y] \mid \forall y.T_1[x,y] \ . \end{aligned} \tag{4.19}$$

It is easy to show that this fragment defines the same set of formulas as (4.14), however we will rather work with definition (4.19) in order to demonstrate generality of our approach.

A nice property of definition (4.19), as well as combinational scheme (4.16) in general, is that it gives a possibility to reuse all steps carried out for **CNF**-transformation for first-order formulas. In particular, it is easy to obtain the result of **NNF**-transformation for combination of fragments having their **NNF**-definitions as follows:

$$
\begin{array}{llcl}
[\mathcal{GF}|\mathcal{FO}^2]^{nnf} & ::= & (\neg)A \ \mid \ F_1 \ \mid \ T_1 \ . & \\
[\mathcal{GF}']^{nnf} & ::= & P \ \mid \ F_1 \mathbin{\wedge\kern-0.6em\vee} F_2 \ \mid \ \forall\overline{y}.[G \rightarrow F_1] \mid \ \exists\overline{y}.[G \wedge F_1] \ . & (4.20) \\
[\mathcal{FO}^{2'}]^{nnf} & ::= & P[x,y] \ \mid \ T_1[x,y] \mathbin{\wedge\kern-0.6em\vee} T_2[x,y] \ \mid \ Qy.T_1[x,y] \ . &
\end{array}
$$

The *structural transformation* for the combined fragment can be also obtained by *joining* those for their respective parts: $[P]^{str}_{gt} := \mathtt{P}_P \wedge [P]^{def}_{gt}$, where:

$$
\begin{array}{rcll}
[P]^{def}_{gt} := [(\neg)A]^{def}_{gt} & = & \forall\overline{x}.(\mathtt{P}_F \rightarrow (\neg)A) & \mid \\
{[F_1]^{def}_{gt}} & = & [F_1]^{def}_{g} & \mid \\
{[T_1]^{def}_{gt}} & = & [T_1]^{def}_{t} \ . &
\end{array}
$$

$$
\begin{array}{rcll}
[F]^{def}_{g} := [F_1 \mathbin{\wedge\kern-0.6em\vee} F_2]^{def}_{g} & = & \forall\overline{x}.(\mathtt{P}_F \rightarrow \mathtt{P}_{F_1} \mathbin{\wedge\kern-0.6em\vee} \mathtt{P}_{F_2}) \wedge [F_1]^{def}_{g} \wedge [F_2]^{def}_{g} & \mid \\
{[\forall\overline{x}.[G \rightarrow F_1]]^{def}_{g}} & = & \forall\overline{x}.(\mathtt{P}_F \rightarrow \forall\overline{y}.(G \rightarrow \mathtt{P}_{F_1})) \wedge [F_1]^{def}_{g} & \mid \\
{[\exists\overline{x}.[G \wedge F_1]]^{def}_{g}} & = & \forall\overline{x}.(\mathtt{P}_F \rightarrow \exists\overline{y}.(G \wedge \mathtt{P}_{F_1})) \wedge [F_1]^{def}_{g} & \mid \\
{[P]^{def}_{g}} & = & [P]^{def}_{gt} \ . &
\end{array}
$$

$$
\begin{array}{rcll}
[T]^{def}_{t} := [T_1[x,y] \mathbin{\wedge\kern-0.6em\vee} T_2[x,y]]^{def}_{t} & = & \forall xy.(\mathtt{P}_T \rightarrow \mathtt{P}_{T_1} \mathbin{\wedge\kern-0.6em\vee} \mathtt{P}_{T_2}) \wedge [T_1]^{def}_{t} \wedge [T_2]^{def}_{t} & \mid \\
{[Qy.T_1[x,y]]^{def}_{t}} & = & \forall x.(\mathtt{P}_T \rightarrow Qy.\mathtt{P}_{T_1}) \wedge [T_1]^{def}_{t} & \mid \\
{[P]^{def}_{t}} & = & [P]^{def}_{gt} \ . &
\end{array}
$$

Here we have pushed the base case for P down in order to avoid possible loops (recall, that the definition for the last case is fired only when all previous cases do not apply). Well, it remains to add nothing but to say that the **CNF**-transformation for the resulted formulas will map them to a clause class that is a *union* of clause classes (**G**) and (**T**) defined for the guarded and two-variable formulas in previous sections.

Saturation of the clause set

In order to decide the union of clause classes (**G**) and (**T**), we should consider all possible inferences between clauses from these classes. However, since all inferences within each class have been already considered and proven to preserve them, it suffices to enumerate inferences that involve only clauses from *different* clause classes.[3] And indeed, it is possible to show that all inferences between (**G**) and (**T**) belong to either from these fragments: see Table 4.10. This proves the following lemma:

[3] It is important that our saturation strategies are *compatible*, i.e., there are common parameters that can be used for *both* saturation procedures. For our procedures this is indeed the case, where

Table 4.10 Possible inferences between the clauses of the guarded and two-variable fragments

$\mathtt{T.1}\colon \Rightarrow$	$:\mathtt{G.1}$	$\mathtt{OR}[\mathtt{G.2.1.1};\mathtt{T.3.1.1}]\colon \neg\hat{p}[!x] \vee \hat{\alpha}[\hat{f}(\overline{x}),\overline{x}]$	$:\mathtt{G.2}$
$\mathtt{OR}[\mathtt{G.1.1.1};\mathtt{T.2.1.1}]\colon \hat{\alpha}[\hat{c}]$	$:\mathtt{G.1}$	$\mathtt{OR}[\mathtt{T.3.1.1};\mathtt{G.2.1.2}]\colon \neg\hat{p}[!x] \vee \hat{\alpha}[\hat{f}(\overline{x}),\overline{x}]$	$:\mathtt{G.2}$
$\mathtt{OR}[\mathtt{T.2.1.1};\mathtt{G.1.1.2}]\colon \hat{\alpha}[\hat{c}]$	$:\mathtt{G.1}$	$\mathtt{OR}[\mathtt{G.2.1.1};\mathtt{T.4.1.1}]\colon \neg\hat{p}[!x] \vee \hat{\alpha}[\hat{f}(\overline{x}),\overline{x}]$	$:\mathtt{G.2}$
$\mathtt{OR}[\mathtt{G.1.1.1};\mathtt{T.4.1.1}]\colon \hat{\alpha}[\hat{c}]$	$:\mathtt{G.1}$	$\mathtt{OR}[\mathtt{T.4.1.1};\mathtt{G.2.1.2}]\colon \neg\hat{p}[!x] \vee \hat{\alpha}[\hat{f}(\overline{x}),\overline{x}]$	$:\mathtt{G.2}$
$\mathtt{OR}[\mathtt{T.4.1.1};\mathtt{G.1.1.2}]\colon \hat{\alpha}[\hat{c}]$	$:\mathtt{G.1}$	$\mathtt{OR}[\mathtt{T.2.1.1};\mathtt{G.2.2.1}]\colon \hat{\alpha}[x,y]$	$:\mathtt{T.2}$
$\mathtt{OR}[\mathtt{G.2.1.1};\mathtt{T.2.1.1}]\colon \neg\hat{p}[!x] \vee \hat{\alpha}[\hat{f}(\overline{x}),\overline{x}]$	$:\mathtt{G.2}$	$\mathtt{OR}[\mathtt{T.3.1.1};\mathtt{G.2.2.1}]\colon \hat{\alpha}[f(x),x]$	$:\mathtt{T.3}$
$\mathtt{OR}[\mathtt{T.2.1.1};\mathtt{G.2.1.2}]\colon \neg\hat{p}[!x] \vee \hat{\alpha}[\hat{f}(\overline{x}),\overline{x}]$	$:\mathtt{G.2}$	$\mathtt{OR}[\mathtt{T.4.1.1};\mathtt{G.2.2.1}]\colon \hat{\alpha}[x]$	$:\mathtt{T.4}$

Lemma 4.2.1 (Closure of $(\mathbf{G}){\cup}(\mathbf{T})$). *There is a strategy based on ordered resolution that given a set N of clauses from $(\mathbf{G}) \cup (\mathbf{T})$, computes its saturation N', which consists of clauses from $(\mathbf{G}) \cup (\mathbf{T})$ constructed over the signature of N extended possibly with unary predicate symbols of form p_L, where $L = a[x]$, or $L = \neg a[x]$ for some predicate symbol a that occurs in N.*

Complexity

Lemma 4.2.1 implies that the structural combination $\mathcal{GF}|\mathcal{FO}^2$ of the guarded and two-variable fragments, is decidable by resolution. Now we are curious about the complexity of this decision procedure. All calculations and remarks used to estimate the number of clauses generated for the guarded and two variable fragments are valid in our case as well, which gives us a complexity bound:

$$\boxed{\boldsymbol{c},\ \boldsymbol{t} \ = \ 2^{\boldsymbol{n}\cdot 2^{\boldsymbol{w}\cdot(\log \boldsymbol{w}+\epsilon)}} \ + \ 2^{O(\boldsymbol{n})} \ = \ \boxed{2^{\boldsymbol{n}\cdot 2^{\boldsymbol{w}\cdot(\log \boldsymbol{w}+\epsilon)}}}} \tag{4.21}$$

Since our saturation procedure is non-deterministic, this immediatly implies that $\mathcal{GF}|\mathcal{FO}^2$ can be decided in 2NEXPTIME and its bounded-variable variant $\mathcal{GF}^k|\mathcal{FO}^2$ can be decided in NEXPTIME. However, it is possible to refine the first bound after an additional observation:

Observation 1: *The* Splitting *rule is applied only on clauses from* $(\mathbf{T})$. *The maximal number of such clauses is* $2^{O(\boldsymbol{n})}$.

Well, how does this observation help us to obtain a better complexity for $\mathcal{GF}|\mathcal{FO}^2$? Our non-deterministic saturation procedure for $\mathcal{GF}|\mathcal{FO}^2$ can be seen as a computation tree, where on each branch at most $2^{O(\boldsymbol{n})}$ non-deterministic binary choices are made. Every branch of this tree has the length at most $2^{\boldsymbol{n}\cdot 2^{\boldsymbol{w}\cdot(\log \boldsymbol{w}+\epsilon)}}$, so, the

we can use $\mathcal{OR}^{\succ}_{Sel}$ parametrised with a *simple* *CAP*-ordering and a selection function for the guarded clauses as has been defined before size of this binary tree is at most $2^{2^{O(\boldsymbol{n})}} \cdot 2^{\boldsymbol{n} \cdot 2^{\boldsymbol{w} \cdot (\log \boldsymbol{w} + \epsilon)}} \;=\; 2^{2^{O(\boldsymbol{n}) + \boldsymbol{w} \cdot (\log \boldsymbol{w} + \epsilon)}}$. Hence, a saturation procedure with *backtracking* decides satisfiability of $\mathcal{GF}|\mathcal{FO}^2$ in time $\boxed{\boldsymbol{t} \;=\; 2^{2^{O(\boldsymbol{n}) + \boldsymbol{w} \cdot (\log \boldsymbol{w} + \epsilon)}}}$.

Theorem 4.2.2. *There is a resolution-based decision procedure for fragments $\mathcal{GF}|\mathcal{FO}^2$ and $\mathcal{GF}^k|\mathcal{FO}^2$ with complexity 2EXPTIME and NEXPTIME respectively.*

Note that the results stated in Theorem 4.2.2 are optimal, since the lower bounds hold already for the fragments $\mathcal{GF}$ and $\mathcal{FO}^2$ which are contained in $\mathcal{GF}|\mathcal{FO}^2$ and $\mathcal{GF}^k|\mathcal{FO}^2$ respectively.

4.2.2 Deciding Combinations with the Monadic Fragment

In this section we demonstrate how to combine resolution decision procedures presented in the previous section, with the one for the monadic fragment described in subsection 4.1.3. Unfortunately, these combinations are not very straightforward: there are certain difficulties when combining *structural transformations* for the monadic fragment and the other fragments, since we introduce definitional predicates for monadic formulas in the outermost way—not like it is done for the guarded and two-variable fragments. Although it is possible to define the structural transformation in a consistent way w.r.t. the previous decision procedures, we prefer to give an alternative solution that is easier to describe. The idea is to apply the structural transformation in two steps: in step (**1**) subformulas that belong to different fragments are separated, and in step (**2**) structural transformations are applied to each separate part as usual.

Starting from recursive definition (4.18) for fragment $\mathcal{GF}|\mathcal{FO}^2|\mathcal{M}_f$, we first put it into the *negation normal form*, as usual:

$$
\begin{array}{rcl}
[\mathcal{GF}|\mathcal{FO}^2|\mathcal{M}_f]^{nnf} & ::= & (\neg)A \quad | \quad F_1 \quad | \quad T_1 \quad | \quad M_1 \ . \\
[\mathcal{GF}']^{nnf} & ::= & P \quad | \quad F_1 \wedge\!\!\!\!\vee F_2 \quad | \quad \forall\overline{y}.[G \rightarrow F_1] \ | \quad \exists\overline{y}.[G \wedge F_1] \ . \\
[\mathcal{FO}^{2\prime}]^{nnf} & ::= & P[x,y] \quad | \quad T_1[x,y] \wedge\!\!\!\!\vee T_2[x,y] \quad | \quad Qy.T_1[x,y] \ . \\
[\mathcal{M}_f{}']^{nnf} & ::= & P[x] \quad | \quad M_1[x] \cdot \{x/f(x)\} \quad | \ M_1 \wedge\!\!\!\!\vee M_2 \quad | \quad Qx.M_1 \ .
\end{array}
\tag{4.22}
$$

Second, we introduce definitional predicates for subformulas that correspond to cases P of recursive definitions—those subformulas that where "exchanged as atoms" during construction of the combined formula. After this step, our formula should have form:

$$
\mathtt{P}_0 \wedge \bigwedge_{i \in I_g} \forall\overline{x}.(\mathtt{P}^i_g[\overline{x}] \rightarrow F_i[\overline{x}]) \wedge \bigwedge_{i \in I_t} \forall\overline{x}.(\mathtt{P}^i_t[\overline{x}] \rightarrow T_i[\overline{x}]) \wedge \bigwedge_{i \in I_m} \forall\overline{x}.(\mathtt{P}^i_n[\overline{x}] \rightarrow M_i[\overline{x}]) \tag{4.23}
$$

where $\mathtt{P}^i_g$, $\mathtt{P}^i_t$, $\mathtt{P}^i_m$ are definitional predicates introduced respectively for guarded subformulas, two-variable subformulas and monadic subformulas, $\mathtt{P}_0$ is some (of these) definitional predicate, and $F_i \in \mathcal{GF}$, $T_j \in \mathcal{FO}^2$, $M_k \in \mathcal{M}_f$ are respectively guarded, two-variable and monadic formulas. For example, (4.15) is translated to:

$$\begin{aligned} &\mathtt{p}_m \wedge (\mathtt{p}_m \rightarrow \overbrace{\forall \mathtt{x}.[\mathtt{Nat(x)} \rightarrow \mathtt{Nat(s(x))}] \wedge \mathtt{p}_t}^{\in \mathcal{M}_f}) \wedge \\ &\qquad \wedge (\mathtt{p}_t \rightarrow \underbrace{\forall \mathtt{xy}.[\mathtt{Nat(x)} \wedge \mathtt{Nat(y)} \rightarrow \mathtt{p}_g\mathtt{(x,y)}]}_{\in \mathcal{FO}^2}) \wedge \\ &\qquad\qquad \wedge \forall \mathtt{xy}.[\mathtt{p}_g\mathtt{(x,y)} \rightarrow \underbrace{\exists \mathtt{z}.(\mathtt{Sum(x,y,z)} \wedge \mathtt{Nat(z)})}_{\in \mathcal{GF}}] \end{aligned}$$

After this transformation, each individual conjunct is translated to clauses as usual (according to structural transformation for every fragment). In the end, a set of clauses from the union of clause classes (**G**), (**T**) and (**M**) is obtained. By computing all cross-inferences between clause class (**M**) and clause classes (**G**) and (**T**), we conclude that every pair of these classes is closed under our resolution strategy: see Table 4.11. All inferences in this table are fairly straightforward, except that a couple of the following invariants are additionally used:

Table 4.11 Possible inferences between the clauses from (**M**) and from (**G**) ∪ (**T**)

OR[M.1.1, G.2.1.∗]: M.2 OR[M.1.1; G.2.2.1]: M.1 OR[M.1.2; G.1.1.∗]: G.1 OR[M.1.2; G.2.1.∗]: G.2 OR[M.1.2; G.2.2.1]: M.1 OR[M.2.1; G.1.1.∗]: G.1	OR[M.2.1; G.2.1.∗]: G.2 OR[M.2.2; G.1.1.∗]: G.1 OR[M.2.2; G.2.1.∗]: G.2 OR[M.3.1; G.2.1.∗]: G.2 OR[M.4.1; G.2.1.∗]: G.2 OR[M.4.1; G.2.2.1]: G.4	T.3, T.4 ⇔ M.1 OR[M.2.1, T.1.1] : T.1 OR[M.2.1, T.2.1.1]: T.2 OR[M.2.2, T.1.1] : T.1	OR[M.2.2, T.2.1.1]: T.2 OR[M.3.1, T.2.1.1]: M.1 OR[M.4.1, T.2.1.1]: M.4

Invariant 1: *The guard literal in clauses of form* 2 *from* (**G**) *cannot be a definitional predicate* $\mathtt{P}_m$ *for monadic formulas. Hence resolution inferences between clauses of form* 2, 3 *and* 4 *from* (**M**) *and clauses of form* 2.2.1 *from* (**G**) *are not possible.*

Invariant 2: *Clauses from* (**T**) *cannot contain definitional predicates* $\mathtt{P}_m$ *with arity greater than* 2. *Hence a resolution inference between clauses* M.2.1 *and* T.2.1.1 *is possible only if the first clause contains two variables, and hence the result of this inference is a clause of form* 1 *from* (**M**).

The case analysis summarised in Table 4.11 together with calculations analogous to those given in the previous section implies the following result:

Theorem 4.2.3. *There is a resolution-based decision procedures for the following fragments of the indicated complexities: $\mathcal{GF}|\mathcal{FO}^2|\mathcal{M}$: 2EXPTIME, $\mathcal{GF}^k|\mathcal{FO}^2|\mathcal{M}$: NEXPTIME. These decision procedures are theoretically optimal, since the lower bounds hold already for subfragments $\mathcal{GF}$ and $\mathcal{FO}^2$ respectively.*

4.2.3 Undecidability Results

In the previous section we have demonstrated how resolution-based decision procedures for structural combinations of the guarded, two-variable and monadic fragments of first-order logic, can be obtained in a modular way from those for their parts. It is known that all these fragments remain decidable also with equality. Therefore, a natural question is: *do combinations of these fragments remain decidable when equality is allowed?* Unfortunately, the answer is negative for almost all combinations of guarded, two-variable and monadic fragments with equality. More precisely, we show that already combinations $\mathcal{GF}^3_{\simeq}|\mathcal{FO}^2$, $\mathcal{GF}^3|\mathcal{FO}^2_{\simeq}$ and $\mathcal{GF}^3|\mathcal{FO}^2|\mathcal{M}_{\simeq}$ are undecidable.

Our undecidability proof is by a reduction from the satisfiability problem for the *Goldfarb class* (see subsection 3.3.1). For any sentence $F' = \forall xy.\exists z.F$ from the Goldfarb class (where F is quantifier-free), we construct the formula:

$$F_{GT} \;:=\; \underbrace{\forall xy.p_1(x,y)}_{\in\,\mathcal{FO}^2} \;\wedge\; \underbrace{\forall xy.[p_1(x,y)\rightarrow\exists z.p_2(x,y,z)] \;\wedge\; \forall xyz.[p_2(x,y,z)\rightarrow F]}_{\in\,\mathcal{GF}^3}$$

Let F^-_{GT} be obtained from F_{GT} by replacing every occurrence of equality with a fresh binary atom $E(x,y)$ and let $F_E := \forall xy.[E(x,y) \leftrightarrow x \simeq y]$ be the "definition" for $E(x,y)$. Then:

$$\begin{array}{rrl} & (\boldsymbol{i}) & F' \textit{ is (finitely) satisfiable} \\ \textit{iff} & (\boldsymbol{ii}) & F_{GT} \textit{ is (finitely) satisfiable} \\ \textit{iff} & (\boldsymbol{iii}) & F^-_{GT} \wedge F_E \textit{ is (finitely) satisfiable.} \end{array}$$

Note, that F^-_{GT} is a conjunction of formulas from $\mathcal{FO}^2$ and $\mathcal{GF}^3$ *without equality.* Finally, observe that F_E is expressible in every fragment $\mathcal{GF}^2_{\simeq}$, $\mathcal{FO}^2_{\simeq}$ and $\mathcal{FO}^2|\mathcal{M}_{\simeq}$:

$$\underbrace{\forall xy.[E(x,y) \leftrightarrow x \simeq y]}_{\in\,\mathcal{FO}^2_{\simeq}} \;\equiv$$

$$\equiv\; \underbrace{\forall xy.[E(x,y)\rightarrow x \simeq y] \wedge \forall xy.[x \simeq y \rightarrow E(x,y)]}_{\in\,\mathcal{GF}_{\simeq}} \;\equiv$$

$$\equiv\; \underbrace{\forall xy.[E(x,y) \leftrightarrow \overbrace{x \simeq y}^{\in\,\mathcal{M}_{\simeq}}]}_{\in\,\mathcal{FO}^2|\mathcal{M}_{\simeq}}$$

Therefore, formula $F_{GT}^{-} \wedge F_E$ is expressible in *all* fragments $\mathcal{GF}^3_{\simeq}|\mathcal{FO}^2$, $\mathcal{GF}^3|\mathcal{FO}^2_{\simeq}$ and $\mathcal{GF}^3|\mathcal{FO}^2|\mathcal{M}_{\simeq}$. This translation $F' \Rightarrow F_{GT}^{-} \wedge F_E$ provides a reduction from the Goldfarb class.

Theorem 4.2.4. *The fragments $\mathcal{GF}^3_{\simeq}|\mathcal{FO}^2$, $\mathcal{GF}^3|\mathcal{FO}^2_{\simeq}$ and $\mathcal{GF}^3|\mathcal{FO}^2|\mathcal{M}_{\simeq}$ form conservative reduction classes, i.e., they are undecidable for (finite) satisfiability.*

4.3 Paramodulation-based Decision Procedures

In this section we formulate several paramodulation-based decision procedures for extensions of the *guarded fragment* with *equality*. The main principle behind such decision procedures remains the same, except that we need to consider additional rules for equality.

A first saturation-based decision procedure for the guarded fragment with equality $\mathcal{GF}_{\simeq}$, has been formulated in [Ganzinger & de Nivelle, 1999]. Surprisingly, this procedure is quite straightforward and does not require additional simplification rules, in contrast, say, to the *monadic class* [see Bachmair et al., 1993*b*]. We analyse the complexity of this decision procedure and derive essentially the same results as for the guarded fragment without equality, which once again proves the robustness of the guarded fragment w.r.t. its extensions.

Starting from a paramodulation-based decision procedure for $\mathcal{GF}_{\simeq}$ as a basis, we consider further extensions of $\mathcal{GF}_{\simeq}$ with (**1**) constants, (**2**) *functionality* and (**3**) *counting*. These extensions are of special interest, because they allow one to express well-known constructors in description logics, namely *nominals*, *functional restrictions* and *(qualified) number restrictions*. We will see that, although the procedure for $\mathcal{GF}_{\simeq}$ can be fairly easily extended to the case with constants (however with some complexity issues), the guarded fragment with functionality appears to be more difficult and is in general *undecidable* [Grädel, 1999]. Yet we do not give up at this point but consider a *restricted version* of this fragment, where functional atoms may be used as guards only. This fragment is still appropriate for description logics and can be decided using the *ordered paramodulation calculus*.

4.3.1 Guarded Fragment with Equality

Shortly after a resolution-based decision procedure for the *guarded fragment* without equality has been formulated in [de Nivelle, 1998], Ganzinger & de Nivelle [1999] have proposed an extension of this procedure for the case with equality. Their decision procedure is based on the *ordered paramodulation calculus*, an extension of the *ordered resolution calculus* with special inference rules for treatment of equality (see subsection 3.5.3). However, the main principles behind the procedure remained the same: (***i***) *formulate a clause class which contains* **CNF**-*translations for the guarded formulas* and (***ii***) *provide a saturation strategy which respects this clause class.*

In this section we follow the main ideas of Ganzinger & de Nivelle [1999], and extend a resolution-based decision procedure for $\mathcal{GF}$ given in subsection 4.1.1, to the case with equality. We demonstrate, that our version of this procedure gives an optimal (and actually the best known) complexity not only for the guarded fragment $\mathcal{GF}_{\simeq}$, but also for its bounded-variable version $\mathcal{GF}^k_{\simeq}$.

A *guarded fragment with equality* $\mathcal{GF}_{\simeq}$ is defined by the same grammar (3.17) as the one without equality, except that now all atoms (including guards) might be equational atoms. It should not come as a big surprise that all **CNF**-transformation steps described in subsection 4.1.1, remain valid for the case with equality. Hence, a clause class for $\mathcal{GF}_{\simeq}$ is essentially the same as class (**G**) given in Table 4.2, except that equational atoms are allowed in clauses. It turns out, that in order to capture all inferences from the guarded clauses with equality, we need an additional clause type described by clause scheme U in Table 4.12. The clauses of this type can be seen as *semantically guarded* clauses with an implicit guard $(x \simeq x)$: they originate from guarded formulas of form $\forall x.[(x \simeq x) \rightarrow F[x]]$ which are equivalent to a simpler formula $\forall x.F[x]$. Note how we distinguish non-equational atoms from atoms that possibly contain equality: the first are denoted with a parameter p, whereas for the last we use a parameter a.

Table 4.12 A clause class for the guarded fragment with equality

		Clause scheme	*Description*
($\mathbf{G}^{\simeq}$):	1	$\hat{\alpha}[\hat{c}]$	*a ground clause whose functional terms are constants;*
	2	$\neg\hat{a}[!\overline{x}] \vee \hat{\alpha}[\hat{f}(\overline{x}), \overline{x}]$	*a guarded clause with a possibly equational guard;*
	U	$\hat{\alpha}[x]$	*non-functional clauses with one variable.*
		where $a := p \mid \simeq;\quad l := a \mid \neg a;\quad \alpha := \vee\{l\}$	

In order to check satisfiability for a set of clauses from ($\mathbf{G}^{\simeq}$), we employ a general-purpose *ordered paramodulation calculus* $\mathcal{OP}^{\succ}_{Sel}$ formulated in 3.5.3 (see System 4 for new inference rules). Any *simple* ordering (see Definition 4.1.4) that is *admissible for paramodulation*, perfectly suits for our decision procedure (in particular, the one formulated after this definition). The selection strategy remains unchanged, i.e., we set a selection function *Sel* to select the guard literal in non-functional guarded clauses. By applying this strategy to clauses from ($\mathbf{G}^{\simeq}$), we obtain inferences that are summarised in Table 4.13. This table has essentially the same logical structure as the one for the guarded fragment without equality (see Table 4.3), however we have two additional inference rules to match the clauses from ($\mathbf{G}^{\simeq}$) with: the Ordered

Table 4.13 Possible inferences between clauses for the guarded fragment with equality

1	$\hat{\alpha}[\hat{c}]$	
1.1	$\hat{\alpha}[\hat{c}] \vee \boldsymbol{l}[\hat{\boldsymbol{c}}]$	
1.1.1	$\hat{\alpha}[\hat{c}] \vee \underline{\boldsymbol{p}[\hat{\boldsymbol{c}}]}^{\star}$	$:\texttt{OR.1}$
1.1.2	$\hat{\alpha}[\hat{c}] \vee \neg\underline{\boldsymbol{p}[\hat{\boldsymbol{c}}]}$	$:\texttt{OR.2}$
1.1.3	$\hat{\alpha}[\hat{c}] \vee \underline{p[\hat{c}]} \vee \underline{\boldsymbol{p}[\hat{\boldsymbol{c}}]}$	$:[\![\,\texttt{OF}\,]\!]$
1.1.4	$\hat{\alpha}[\hat{c}] \vee \underline{\hat{\boldsymbol{c}}} \simeq \hat{\boldsymbol{c}}^{\star}$	$:\texttt{OP.1}$
1.1.5	$\hat{\alpha}[\underline{c}, \hat{c}]$	$:\texttt{OP.2}$
1.1.6	$\hat{\alpha}[\hat{c}] \vee \underline{\boldsymbol{c}} \not\simeq \underline{\boldsymbol{c}}$	$:\texttt{RR}$
$\texttt{OR}[1.1.1; 1.1.2]$	$:\hat{\alpha}[\hat{c}]$	$:1$
$\texttt{OP}[\mathbf{1.1.4}; \mathbf{1.1.5}]$	$:\hat{\alpha}[\hat{c}] \vee \hat{\alpha}[\hat{c}, \hat{c}]$	$:1$
$\texttt{RR}[\mathbf{1.1.6}]$	$:\hat{\alpha}[\hat{c}]$	$:1$
$\bot$	$\square$	

$\texttt{U}$	$\hat{\alpha}[x]$	
$\texttt{U}$**.1**	$\hat{\alpha}[x] \vee \boldsymbol{l}[!\boldsymbol{x}]$	
$\texttt{U}$**.1.1**	$\hat{\alpha}[x] \vee \underline{\boldsymbol{p}[!\boldsymbol{x}]}^{\star}$	$:\texttt{OR.1}$
$\texttt{U}$**.1.2**	$\hat{\alpha}[x] \vee \neg\underline{\boldsymbol{p}[!\boldsymbol{x}]}$	$:\Rightarrow$ of form 2
$\texttt{U}$**.1.3**	$\hat{\alpha}[x] \vee \underline{\boldsymbol{x}} \not\simeq \underline{\boldsymbol{x}}$	$:\texttt{RR}$
$\texttt{OR}[\texttt{U}.\mathbf{1.1}; \mathbf{1.1.2}]$	$:\hat{\alpha}[\hat{c}] \vee \hat{\alpha}[\hat{c}]$	$:1$
$\texttt{RR}[\texttt{U}.\mathbf{1.3}]$	$:\hat{\alpha}[x]$	$:\texttt{U}$
$\texttt{U}$**.2**	$\hat{\alpha} :\Rightarrow$ of form 1	

2	$\neg\hat{a}[!\overline{x}] \vee \hat{\alpha}[\hat{f}(\overline{x}), \overline{x}]$	
2.1	$\neg\hat{a}[!\overline{x}] \vee \hat{\alpha}[\hat{f}(\overline{x}), \overline{x}] \vee \hat{\boldsymbol{l}}[!\hat{\boldsymbol{f}}(\overline{\boldsymbol{x}}), \overline{\boldsymbol{x}}]$	
2.1.1	$\neg\hat{a}[!\overline{x}] \vee \hat{\alpha}[\hat{f}(\overline{x}), \overline{x}] \vee \underline{\boldsymbol{p}[!\hat{\boldsymbol{f}}(\overline{\boldsymbol{x}}), \overline{\boldsymbol{x}}]}^{\star}$	$:\texttt{OR.1}$
2.1.2	$\neg\hat{a}[!\overline{x}] \vee \hat{\alpha}[\hat{f}(\overline{x}), \overline{x}] \vee \neg\underline{\boldsymbol{p}[!\hat{\boldsymbol{f}}(\overline{\boldsymbol{x}}), \overline{\boldsymbol{x}}]}$	$:\texttt{OR.2}$
2.1.3	$\neg\hat{a}[!\overline{x}] \vee \hat{\alpha}[\hat{f}(\overline{x}), \overline{x}] \vee \underline{p[\hat{f}(\overline{x}), \overline{x}]} \vee \underline{\boldsymbol{p}[!\hat{\boldsymbol{f}}(\overline{\boldsymbol{x}}), \overline{\boldsymbol{x}}]}$	$:\texttt{OF}$
2.1.4	$\neg\hat{a}[!\overline{x}] \vee \hat{\alpha}[\hat{f}(\overline{x}), \overline{x}] \vee \underline{\boldsymbol{f}(\overline{\boldsymbol{x}})} \simeq \{\hat{\boldsymbol{f}}(\overline{\boldsymbol{x}}), \overline{\boldsymbol{x}}\}^{\star}$	$:\texttt{OP.1}$
2.1.5	$\neg\hat{a}[!\overline{x}] \vee \hat{\alpha}[\underline{f(\overline{x})}, \hat{f}(\overline{x}), \overline{x}]$	$:\texttt{OP.2}$
2.1.6	$\neg\hat{a}[!\overline{x}] \vee \hat{\alpha}[\hat{f}(\overline{x}), \overline{x}] \vee \underline{\boldsymbol{f}(\overline{\boldsymbol{x}})} \not\simeq \underline{\boldsymbol{f}(\overline{\boldsymbol{x}})}$	$:\texttt{RR}$
$\texttt{OR}[2.1.1; 2.1.2]$	$:\neg\hat{a}[!\overline{x}] \vee \hat{\alpha}[\hat{f}(\overline{x}), \overline{x}]$	$:2$
$\texttt{OF}[2.1.3]$	$:\neg\hat{a}[!\overline{x}] \vee \hat{\alpha}[\hat{f}(\overline{x}), \overline{x}] \vee p[!\hat{f}(\overline{x}), \overline{x}]$	$:2$
$\texttt{OP}[\mathbf{2.1.4}; \mathbf{2.1.5}]$	$:\neg\hat{a}[!\overline{x}] \vee \hat{\alpha}[\hat{f}(\overline{x}), \overline{x}]$	$:2$
$\texttt{RR}[\mathbf{2.1.6}]$	$:\neg\hat{a}[!\overline{x}] \vee \hat{\alpha}[\hat{f}(\overline{x}), \overline{x}]$	$:2$
$\texttt{OR}[\texttt{U}.1.1; 2.1.2]$	$:\hat{\alpha}[\hat{f}(\overline{x}), \overline{x}] \vee \neg\hat{a}[!\overline{x}] \vee \hat{\alpha}[\hat{f}(\overline{x}), \overline{x}]$	$:2$
2.2	$\neg\hat{\boldsymbol{a}}[!\overline{\boldsymbol{x}}]^{\sharp} \vee \hat{\alpha}[\overline{x}]$	$:\mathit{Sel}$
2.2.1	$\neg\underline{\hat{\boldsymbol{a}}[!\overline{\boldsymbol{x}}]} \vee \hat{\alpha}[\overline{x}]$	$:\texttt{OR.2}$
2.2.2	$\underline{\boldsymbol{x_1}} \not\simeq \underline{\boldsymbol{x_2}} \vee \hat{\alpha}[x_1, x_2]$	$:\texttt{RR}$
$\texttt{OR}[1.1.1; 2.2.1]$	$:\hat{\alpha}[\hat{c}]$	$:1$
$\texttt{OR}[2.1.1; 2.2.1]$	$:\neg\hat{a}[!\overline{x}] \vee \hat{\alpha}[\hat{f}(\overline{x}), \overline{x}]$	$:2$
$\texttt{RR}[\mathbf{2.2.2}]$	$:\hat{\alpha}[x]$	$:\texttt{U}$
$\texttt{OR}[\texttt{U}.\mathbf{1.1}; \mathbf{2.2.1}]$	$:\hat{\alpha}[x] \vee \hat{\alpha}[x]$	$:\texttt{U}$

Paramodulation rule (shortly OP) and the Reflexivity Resolution (short RR). New cases which correspond to these inference rules are indicated with bold numbers.

For the clauses of type 1 from $(\mathbf{G}^{\simeq})$, we get the following new cases: case 1.1.4 that matches a clause to the left premise of the Ordered Paramodulation rule, case 1.1.5 that matches a clause to the right premise of this rule, and case 1.1.6 that matches a clause to the premise of the Reflexivity Resolution rule. For clauses of types 2 and U, we have similar cases. Note that certain cases are not possible: for example, it is not possible to use a clause of form 2.2 as the right premise of the Ordered Paramodulation rule, since paramodulation *into a variable* is not allowed by condition (vi) of this rule (see System 4), but all subterms of such clause are variables. The same holds for clauses of type U. Note that clauses of these types appear from application of Reflexivity Resolution rule to clauses of type 2.2 with equational guards.

It remains, perhaps, to give some comments on cases 2.1.4 and 2.1.5. Recall, that according to our scheme notation (see Sections B), expression $f(\overline{x}) \simeq \{\hat{f}(\overline{x}), \overline{x}\}$ represents an atom of form $f(\overline{x}) \simeq g(\overline{x})$ or of form $f(\overline{x}) \simeq x$, where $x \in \overline{x}$. Case 2.1.5 is related to the *simultaneous paramodulation* strategy that is described in Figure 3.3 on p. 87. In fact, we will use this refinement of the Ordered Paramodulation rule in our inferences. So, case 2.1.5 indicates that paramodulation takes place into *every* occurrence of term $f(\overline{x})$ in a clause (*not only* into *one* occurrence in the eligible literal). As will be seen in a moment, this will let us obtain the optimal complexity results for our decision procedure.

The case analysis given in Table 4.13 proves the following lemma:

Lemma 4.3.1 (Closure of $(\mathbf{G}^{\simeq})$)**.** *Let N be a set of clauses of form $(\mathbf{G}^{\simeq})$ and N' be obtained from N by a saturation under $\mathcal{OP}^{\succ}_{\mathrm{Sel}}$ based on a simple order $\succ$ and a selection function Sel that selects a guard in non-functional guarded clauses. Let $\boldsymbol{w}$ be the maximal number of different variables in a clause from N. Then all clauses from N' belong to $(\mathbf{G}^{\simeq})$ and have at most $\boldsymbol{w}$ different variables.*

It turns out (see Appendix C.2.1) that the complexity of our decision procedure is the same as for the case without equality, which implies the analogous result:

Theorem 4.3.2. *There is a paramodulation-based decision procedure for the guarded fragment with equality $\mathcal{GF}_{\simeq}$ which can be implemented in 2EXPTIME. This procedure decides its bounded-variable version $\mathcal{GF}^{k}_{\simeq}$ in EXPTIME.*

4.3.2 Guarded Fragment with Constants

A guarded fragment with equality $\mathcal{GF}_{\simeq}$ alone, seems to be not particularly useful for reasoning in modal and description logics: there are only few constructors that could be expressed in $\mathcal{GF}_{\simeq}$: Ganzinger & de Nivelle [1999] gave an example of *difference logic*—a modal logic where $\Diamond A$ means that A should hold in some *different* world than the current one (this can be expressed in $\mathcal{GF}_{\simeq}$). A real advantage of equality can be gained in combination with other extensions of the guarded fragment.

If we look at the description logic constructors listed in Table 3.2 on p.67, we find several constructors that translate to the first-order logic with equality, namely *(qualified) number restrictions*, *functional restrictions* and *nominals*. Consider the last constructor from these, which presumably, is the simplest one. *Nominals* can be expressed in the other way, as we did for extensions of $\mathcal{EL}$ in subsection 2.4.3, using *constants*: see Table 2.15. Translations for nominals given in this table, belong to the guarded fragment with *equality* and *constants*, in particular the formula $\forall x.(A(x) \rightarrow x \simeq a)$ which expresses an inclusion axiom $A \sqsubseteq \{a\}$. In this section, we are concerned with a decision procedure for this extension of the guarded fragment.

The fact that constants may be admitted in atoms of guarded formulas has been pointed out by Grädel [1999]. He also presented a simple transformation, using which a guarded formula with constants can be transformed into another guarded formula *without constants* in a satisfiability-preserving way.

Elimination of constants

A procedure that eliminates constants from guarded formulas can be described in two steps:

Step 1 *Given a guarded formula $F[c_1,..,c_k]$ containing k constants, pick k fresh variable names $z_1,..,z_k$, and replace all occurrences of constants in this formula with the respective variable: $F[c_1/z_1,..,c_k/z_k]$. This transformation preserves satisfiability of formulas (which can be demonstrated using Skolemization). Note that the resulted formula might not be guarded, since guards do not necessarily contain all constants: for example the formula for nominals above is translated to $\forall x.(A(x) \rightarrow x \simeq z)$, which is not guarded.*

Step 2 *To make a formula guarded again, we put all new variables $z_1,..,z_k$ as* new arguments *of relational predicates. That is, we expand every atom $p(x_1,..,x_n)$ in F (note that only variables are left in arguments), to $p(z_1,..,z_k,x_1,..,x_n)$.*[4] *This transformation also preserves satisfiability of a first-order formula, provided that variables $z_1,..,z_k$ occur free in F (this can be also demonstrated using Skolemization followed by a suitable renaming of predicate symbols). After this step, a formula obtained in Step 1 becomes guarded, since all guards will contain all variables $z_1,..,z_k$.*

Note that transformation in Step 2 does not expand the *equational predicates*: the formula $\forall x.(A(x) \rightarrow x \simeq z)$ from our example is translated into $\forall x.(A(z,x) \rightarrow x \simeq z)$. This means that if F contained *equational* guards, this transformation might still produce non-guarded clauses: for example, the guarded formula

$$\forall x.[(x \simeq a) \rightarrow \forall y.[(x \simeq y) \rightarrow A(x,y)]]$$

is transformed to the non-guarded formula:

$$\forall x.[(x \simeq z) \rightarrow \forall y.[(x \simeq y) \rightarrow A(z,x,y)]]$$

It seems to be that Grädel [1999] did not allow for equational atoms in guard positions (although this is not explicitly stated). Luckily, it is possible to eliminate all equational guards from every guarded formula using an additional transformation: note that the last formula is simply equivalent to $A(z,z,z)$.

Before we continue, we just point out that there is an interesting connection between the above transformation that eliminates constants, and automata/tableau-based decision procedure for DL $\mathcal{ALCIO}$ [Sattler & Vardi, 2001; Horrocks & Sattler, 2001], where nominals are *memorised* in *global states* of an automaton or tableau in a similar way as the new variables in atoms are used.

[4] $p(x_1,..,x_n)$ must be expanded even if it already contains variables $z_1,..,z_k$ among $x_1,..,x_n$

Elimination of equational guards

We describe a procedure that given a guarded formula, possibly containing equational guards, produces another guarded formula without equational guards. Our procedure consists of two steps similar to those described above for elimination of constants. In the first step, we eliminate equational guards, but this might result in a non-guarded formula. In the second step we make our formula guarded again.

Step 1 *Given a guarded formula $F \in \mathcal{GF}_{\simeq}$ (possibly containing constants), we perform the following replacements for its subformulas that are guarded with equational guards:*

$$\begin{array}{rcl} \forall xy.[(x \simeq y) \to F_1[x,y]] & \dashrightarrow & \forall x.F_1[x,y/x]; \\ \forall y.[(x \simeq y) \to F_1[x,y]] & \dashrightarrow & F_1[x,y/x]; \\ \forall x.[(x \simeq c) \to F_1[x]] & \dashrightarrow & F_1[x/c]; \end{array} \tag{4.24}$$

It is easy to see that this transformation produces an equivalent formula from the following fragment:

$$\mathcal{GF}^*_{\simeq} \ ::= \ A \mid \neg F_1 \mid F_1 \wedge F_2 \mid \forall \overline{y}.[G \to F_1] \mid \forall x.F_1[x] \ . \tag{4.25}$$

where G are non-equational guards (possibly with constants).

Step 2 *A resulted formula F' of form* (4.25) *might not be guarded because of the last case. We introduce new guards for such subformulas using a fresh unary atom $e(x)$, which is read as "x is an element of a domain". Using this atom, we* "shield" *every quantified variable from F' as follows:*

$$\begin{array}{rcl} \forall y_1..y_n.[G \to F_1] & \dashrightarrow & \forall y_1..y_n.[G \wedge e(y_1) \wedge \cdots \wedge e(y_n) \to F_1]; \\ \forall x.F_1[x] & \dashrightarrow & \forall x.[e(x) \to F_1[x]]; \end{array} \tag{4.26}$$

and add conjuncts $e(c)$ for every constant c in F', or a conjunct $\exists x.e(x)$ if there is are no constants. For example, the formula $\forall x.[(\forall y.a(x,y) \to y \simeq c) \wedge b(x)]$ of form (4.25) *is translated to: $\forall x.[e(x) \to [\forall y.a(x,y) \wedge e(y) \to y \simeq c) \wedge b(x)]] \wedge e(c)$. We claim that a formula F'' resulted after this transformation from F', is satisfiable iff F' is satisfiable. Indeed, every model of F' can be expanded to a model of F'' by defining $e(x)$ to hold on all elements of the model. Conversely, every model of F'' induces a model of F' by restricting the domain to those elements on which $e(x)$ holds (this set is non-empty because of the additional conjuncts). Needless to say that the result of this transformation is a guarded formula without equational guards.*

By combining elimination of equational guards followed with Grädel's [1999] elimination of constants, we prove the following proposition:

Proposition 4.3.3 (Elimination of Constants from $\mathcal{GF}_{\simeq}$)**.** *There is a polynomial transformation that given a guarded formula $F \in \mathcal{GF}_{\simeq}$ with constants, produces an equisatisfiable guarded formula $F' \in \mathcal{GF}_{\simeq}$ without constants.*

This proposition implies that we can employ our paramodulation-based decision procedure described subsection 4.3.1 for deciding the guarded fragment *with constants*.

Corollary 4.3.4. *There is a paramodulation-based decision procedure for the guarded fragment with equality and constants which can be implemented in 2EXPTIME.*

However a decision procedure through the transformations above does not seem to give (at least directly) a better complexity for the *bounded-variable* guarded fragment with constants, since elimination of constants increases the *width* of the guarded formula on the number of constant symbols in the formula. Moreover, the exact complexity of the bounded-variable guarded fragment with constants seems to be not known so far.

We stress that the bounded-variable guarded fragment, and not the full guarded fragment, is of particular interest for non-classical logics, since problems in modal and description logics correspond to *bounded-variable* first-order formulas.

In this section we demonstrate that $\mathcal{GF}^k_{\simeq}$ remains EXPTIME-complete even with constants using two paramodulation-based decision procedures. The first procedure refines the clause class ($\mathbf{G}^{\simeq}$) defined in subsection 4.3.1, to capture the clauses that are obtained after the elimination of constants described above. The second procedure does not use elimination of constants but extends clause class ($\mathbf{G}^{\simeq}$) to capture guarded formulas with constants directly.

A paramodulation-based procedure employing elimination of constants

Table 4.14 A clause class for the guarded fragment with eliminated constants

	Clause scheme	***Description***
($\mathbf{G}^{\overline{z}}$):	1 $\hat{\alpha}(\boldsymbol{v}, \{\hat{c}\}) \vee \{(\simeq \mid \not\simeq)[\hat{c}]\}$	*a ground clause whose non-equational literals have a fixed (within the clause) prefix of constants $\boldsymbol{v}$ of length $\boldsymbol{k}$;*
	2 $\neg\hat{p}(\overline{z}, \{!\overline{x}\}) \vee$ $\vee\, \hat{\alpha}(\overline{z}, \{\hat{f}(\overline{z}, \overline{x}), \overline{x}\}) \vee$ $\vee\, \{(\simeq \mid \not\simeq)[\hat{f}(\overline{z}, \overline{x}), \overline{x}]\}$	*a guarded clause with non-equational guards whose non-equatioal predicates are prefixed with a fixed (within the clause) variable-vector $\overline{z}$ of length $\boldsymbol{k}$.*
	where $\boldsymbol{v} := \{\hat{c}\}$; $a := p \mid \simeq$; $l := a \mid \neg a$; $\alpha := \vee\{l\}$; $\lvert\boldsymbol{v}\rvert = \lvert\overline{z}\rvert = \boldsymbol{k}$	

Let $F \in \mathcal{GF}_{\simeq}$ be a guarded formula with constants and $F' \in \mathcal{GF}_{\simeq}$ be obtained from F by elimination of constants. Recall that every non-equational atom in F'

Table 4.15 A decision procedure for the guarded fragment with constants through Grädel's [1999] elimination of constants

$\boldsymbol{v} := \{\hat{c}\}$: *a fixed sequence of constants within a clause*

1	$\hat{\alpha}(\boldsymbol{v}, \{\hat{c}\}) \vee \{(\simeq\mid\not\simeq)[\hat{c}]\}$	:The cases are analogous to the those in Table 4.13
2	$\neg\hat{p}(\overline{z}, \{!\overline{x}\}) \vee \hat{\alpha}(\overline{z}, \{\hat{f}(\overline{z}, \overline{x}), \overline{x}\}) \vee \{(\simeq\mid\not\simeq)[\hat{f}(\overline{z}, \overline{x}), \overline{x}]\}$	
2.1	$\neg\hat{p}(\overline{z}, \{!\overline{x}\}) \vee \cdots \vee \boldsymbol{\hat{l}[\overline{z}, !\hat{f}(\overline{z}, \overline{x}), \overline{x}]}$	
2.1.1	$\neg\hat{p}(\overline{z}, \{!\overline{x}\}) \vee \cdots \vee \underline{\boldsymbol{p[\overline{z}, !\hat{f}(\overline{z}, \overline{x}), \overline{x}]}}^{\star}$	:OR.1
2.1.2	$\neg\hat{p}(\overline{z}, \{!\overline{x}\}) \vee \cdots \vee \neg\underline{\boldsymbol{p[\overline{z}, !\hat{f}(\overline{z}, \overline{x}), \overline{x}]}}$	:OR.2
2.1.3	$\neg\hat{p}(\overline{z}, \{!\overline{x}\}) \vee \cdots \vee \underline{p[\overline{z}, \hat{f}(\overline{z}, \overline{x}), \overline{x}]} \vee \underline{\boldsymbol{p[\overline{z}, !\hat{f}(\overline{z}, \overline{x}), \overline{x}]}}$	:OF
2.1.4	$\neg\hat{p}(\overline{z}, \{!\overline{x}\}) \vee \cdots \vee \underline{\boldsymbol{f(\overline{z}, \overline{x})}} \simeq \boldsymbol{\{\hat{f}(\overline{z}, \overline{x}), \overline{x}\}}^{\star}$	:OP.1
2.1.5	$\neg\hat{p}(\overline{z}, \{!\overline{x}\}) \vee \hat{\alpha}(\overline{z}, \{\underline{f(\overline{z}, \overline{x})}, \hat{f}(\overline{z}, \overline{x}), \overline{x}\}) \vee \{(\simeq\mid\not\simeq)[\underline{f(\overline{z}, \overline{x})}, \hat{f}(\overline{z}, \overline{x}), \overline{x}]\}$	:OP.2
2.1.6	$\neg\hat{p}(\overline{z}, \{!\overline{x}\}) \vee \cdots \vee \underline{\boldsymbol{f(\overline{z}, \overline{x})}} \not\simeq \underline{\boldsymbol{f(\overline{z}, \overline{x})}}$	:RR
OR[2.1.1; 2.1.2]:	$\neg\hat{p}(\overline{z}, \{!\overline{x}\}) \vee \cdots$	:2
OF[2.1.3] :	$\neg\hat{p}(\overline{z}, \{!\overline{x}\}) \vee \cdots \vee p[\overline{z}, !\hat{f}(\overline{z}, \overline{x}), \overline{x}]$	:2
OP[2.1.4; 2.1.5]:	$\neg\hat{p}(\overline{z}, \{!\overline{x}\}) \vee \hat{\alpha}(\overline{z}, \{\hat{f}(\overline{z}, \overline{x}), \overline{x}\}) \vee \{(\simeq\mid\not\simeq)[\hat{f}(\overline{z}, \overline{x}), \overline{x}]\}$	:2
RR[2.1.6] :	$\neg\hat{p}(\overline{z}, \{!\overline{x}\}) \vee \cdots$	:2
2.2	$\boldsymbol{\neg\hat{p}(\overline{z}, \{!\overline{x}\})}^{\sharp} \vee \hat{\alpha}(\overline{z}, \{\overline{x}\})$	:*Sel*
2.2.1	$\neg\underline{\boldsymbol{\hat{p}(\overline{z}, \{!\overline{x}\})}} \vee \hat{\alpha}(\overline{z}, \{\overline{x}\})$	:OR.2
OR[1.1.1; 2.2.1]:	$\hat{\alpha}(\boldsymbol{v}, \{\hat{c}\}) \vee \{(\simeq\mid\not\simeq)[\hat{c}]\}$	:1
OR[2.1.1; 2.2.1]:	$\neg\hat{p}(\overline{z}, \{!\overline{x}\}) \vee \hat{\alpha}(\overline{z}, \{\hat{f}(\overline{z}, \overline{x}), \overline{x}\}) \vee \{(\simeq\mid\not\simeq)[\hat{f}(\overline{z}, \overline{x}), \overline{x}]\}$	:2

has a form $a(\overline{z}, \overline{x})$, where $\overline{z}$ is a fixed vector of variables of length $\boldsymbol{k} :=$ the number of constants in F, and $\overline{x}$ contains at most $\boldsymbol{w} := \mathit{width}(F)$ of *other* different variables. By applying the usual **CNF**-transformation, it is easy to show that all resulted clauses belong to class $(\mathbf{G}^{\overline{z}})$ defined in Table 4.14. The number of such clauses is $O(\boldsymbol{n})$, where $\boldsymbol{n} := |F|$, since the number of subformulas in F' is $O(\boldsymbol{n})$ (the only new subformulas in F' are $e(z)$ that are introduced for every variable and constant in F at most once). Now, if we restrict the case analysis from Table 4.13 to clause class $(\mathbf{G}^{\overline{z}})$, we obtain inferences that are sketched in Table 4.15. Note that there is no case U, since we have eliminated non-equational atoms. This case analysis shows that the clause class $(\mathbf{G}^{\overline{z}})$ is closed under paramodulation inferences:

Lemma 4.3.5 (Closure of $(\mathbf{G}^{\simeq})$)**.** *Let N be a set of clauses of form $(\mathbf{G}^{\overline{z}})$ where the length of every prefix $\boldsymbol{v}$ and $\overline{z}$ is $\boldsymbol{k}$, and let $\boldsymbol{w}$ be the maximal number of different constants (variables) in each clause from* 1 *(*2*) from N that do not belong to $\boldsymbol{v}$ $(\overline{z})$. Let N' be obtained from N by a saturation under $\mathcal{OP}^{\succ}_{Sel}$ based on a simple order $\succ$ and a selection function Sel that selects a guard in non-functional guarded clauses. Then each clause from N' belongs to $(\mathbf{G}^{\overline{z}})$ and have the same properties as mentioned for N.*

The refined clause class gives us an optimal complexity also for the bounded-variable version of $\mathcal{GF}$ with constants: $\boxed{\boldsymbol{t} \ \leq \ 2^{\boldsymbol{n}\cdot(2\boldsymbol{w}+\boldsymbol{k})^{(\boldsymbol{w}+\epsilon)}}}$ (see Appendix C.2.2):

Theorem 4.3.6. *There is a paramodulation-based decision procedure for the guarded fragment with equality and constants which can be implemented in 2EXPTIME. This procedure decides the bounded-variable guarded fragment with equality and constants in EXPTIME.*

Note that the paramodulation-based procedure has remained the same—we have just refined the case analysis of inferences to see that it gives an optimal complexity result for the bounded-variable case as well.

A direct paramodulation-based decision procedure

Although the decision procedure described above gives optimal complexity results for the full and the bounded-variable guarded fragments with constants, it is always a good idea to refrain, if possible, from using additional transformations. It turns out that a decision procedure of the same complexity, can be obtained directly, without elimination of constants.

It is easy to observe, that without elimination of constants we obtain a clause class that is similar to ($\mathbf{G}$), except that literals might contain additional constants. This clause class can be captured by clause schemes ($\mathbf{G}^c$) defined in Table 4.16. Intuitively, every clause from ($\mathbf{G}^c$) is obtained from clauses in ($\mathbf{G}^{\simeq}$) by substituting constants for some variables. Note that we do not allow equational guards, assuming that they have been eliminated. By repeating essentially the same case analysis as for $\mathcal{GF}_{\simeq}$ given in Table 4.13, we can prove that clause class ($\mathbf{G}^c$) is closed under paramodulation inferences. This case analysis is summarised in Table 4.17.

Table 4.16 A clause class for the guarded fragment with constants

		Clause scheme	***Description***
($\mathbf{G}^c$):	1	$\hat{\alpha}[\hat{c}]$	*a ground whose arguments are constants;*
	2	$\neg\hat{p}[!\overline{x}, \hat{c}] \vee \hat{\alpha}[\hat{f}(\boldsymbol{v}), \overline{x}, \hat{c}]$	*a clause with non-equational guards which may contain constants, such that every non-constant functional symbol has a fixed sequence of arguments consisting of all variables of the clause and constants.*
		where $\boldsymbol{v} := \{!\overline{x}, \hat{c}\}$; $a := p \mid \simeq$; $l := a \mid \neg a$; $\alpha := \vee\{l\}$;	

To make everything work fine, we need an additional assumption about the ordering $\succ$ and constants. We assume that $(\boldsymbol{i})$ $p(s_1,..,s_n) \succ (c_1 \simeq c_2)$ for every non-equational predicate symbol p and every pair of constants c_1, c_2, and $(\boldsymbol{ii})$ $f(t_1,..,t_m) \succ c$ for every non-constant functional symbol f and every constant c. Any simple *LPO*-ordering can be easily adjusted to fulfil this property by using a precedence

Table 4.17 A direct decision procedure for the guarded fragment with constants

$\boldsymbol{v} := \{!\overline{x}, \hat{c}\}$: *a fixed sequence of variables and constants within a clause*

1	$\hat{\alpha}[\hat{c}]$	
1.1	$\hat{\alpha}[\hat{c}] \vee \boldsymbol{l}[\hat{\boldsymbol{c}}]$	
1.0	$\llbracket \hat{\alpha}[\hat{c}] \vee \hat{\boldsymbol{c}} \simeq \hat{\boldsymbol{c}}^{\star} \rrbracket$	:SP
1.1.1	$\hat{\alpha}[\hat{c}] \vee \underline{\boldsymbol{p}[\hat{\boldsymbol{c}}]}^{\star}$	:OR.1
1.1.2	$\hat{\alpha}[\hat{c}] \vee \neg\underline{\boldsymbol{p}[\hat{\boldsymbol{c}}]}$	:OR.2
1.1.4	$\underline{\hat{\boldsymbol{c}}} \simeq \hat{\boldsymbol{c}}^{\star}$	:OP.1
1.1.5	$\llbracket \hat{\alpha}[\underline{c}, \hat{c}] \rrbracket$	:OP.2
1.1.6	$\hat{\alpha}[\hat{c}] \vee \underline{\boldsymbol{c}} \not\simeq \underline{\boldsymbol{c}}$	:RR
	OR[1.1.1; 1.1.2]: $\hat{\alpha}[\hat{c}]$	:1
	OP[1.1.4; 1.1.5]: $\hat{\alpha}[\hat{c}, \hat{c}]$	:1
	RR[1.1.6] : $\hat{\alpha}[\hat{c}]$	:1
$\perp$	$\square$	

2	$\neg\hat{p}[!\overline{x}, \hat{\boldsymbol{c}}] \vee \hat{\alpha}[\hat{f}(\boldsymbol{v}), \overline{x}, \hat{\boldsymbol{c}}]$	
2.1	$\neg\hat{p}[!\overline{x}, \hat{\boldsymbol{c}}] \vee \hat{\alpha}[\hat{f}(\boldsymbol{v}), \overline{x}, \hat{\boldsymbol{c}}] \vee \hat{\boldsymbol{l}}[!\hat{\boldsymbol{f}}(\boldsymbol{v}), \overline{\boldsymbol{x}}, \hat{\boldsymbol{c}}]$	
2.1.1	$\neg\hat{p}[!\overline{x}, \hat{\boldsymbol{c}}] \vee \hat{\alpha}[\hat{f}(\boldsymbol{v}), \overline{x}, \hat{\boldsymbol{c}}] \vee \underline{\boldsymbol{p}[!\hat{\boldsymbol{f}}(\boldsymbol{v}), \overline{\boldsymbol{x}}, \hat{\boldsymbol{c}}]}^{\star}$	:OR.1
2.1.2	$\neg\hat{p}[!\overline{x}, \hat{\boldsymbol{c}}] \vee \hat{\alpha}[\hat{f}(\boldsymbol{v}), \overline{x}, \hat{\boldsymbol{c}}] \vee \neg\underline{\boldsymbol{p}[!\hat{\boldsymbol{f}}(\boldsymbol{v}), \overline{\boldsymbol{x}}, \hat{\boldsymbol{c}}]}$	:OR.2
2.1.3	$\neg\hat{p}[!\overline{x}, \hat{\boldsymbol{c}}] \vee \hat{\alpha}[\hat{f}(\boldsymbol{v}), \overline{x}, \hat{\boldsymbol{c}}] \vee \underline{p[\hat{f}(\overline{x})..]} \vee \underline{\boldsymbol{p}[!\hat{\boldsymbol{f}}(\boldsymbol{v}), \overline{\boldsymbol{x}}, \hat{\boldsymbol{c}}]}$	:OF
2.1.4	$\neg\hat{p}[!\overline{x}, \hat{\boldsymbol{c}}] \vee \hat{\alpha}[\hat{f}(\boldsymbol{v}), \overline{x}, \hat{\boldsymbol{c}}] \vee \underline{\boldsymbol{f}(\boldsymbol{v})} \simeq \{\hat{\boldsymbol{f}}(\boldsymbol{v}), \overline{\boldsymbol{x}}, \hat{\boldsymbol{c}}\}^{\star}$	:OP.1
2.1.5	$\neg\hat{p}[!\overline{x}, \hat{\boldsymbol{c}}] \vee \hat{\alpha}[\underline{f(\boldsymbol{v})}, \hat{f}(\boldsymbol{v}), \overline{x}, \hat{\boldsymbol{c}}]$	:OP.2
2.1.5′	$\llbracket \neg\hat{p}[!\overline{x}, \hat{\boldsymbol{c}}] \vee \hat{\alpha}[\hat{f}(\boldsymbol{v}), \overline{x}, \underline{c}, \hat{\boldsymbol{c}}] \rrbracket$	:OP.2
2.1.6	$\neg\hat{p}[!\overline{x}, \hat{\boldsymbol{c}}] \vee \hat{\alpha}[\hat{f}(\boldsymbol{v}), \overline{x}, \hat{\boldsymbol{c}}] \vee \underline{\boldsymbol{f}(\boldsymbol{v})} \not\simeq \underline{\boldsymbol{f}(\boldsymbol{v})}$	:RR
	OR[2.1.1; 2.1.2] : $\neg\hat{p}[!\overline{x}, \hat{\boldsymbol{c}}] \vee \hat{\alpha}[\hat{f}(\boldsymbol{v}), \overline{x}, \hat{\boldsymbol{c}}]$	:2
	OF[2.1.3] : $\neg\hat{p}[!\overline{x}, \hat{\boldsymbol{c}}] \vee \hat{\alpha}[\hat{f}(\boldsymbol{v}), \overline{x}, \hat{\boldsymbol{c}}] \vee p[!\hat{f}(\boldsymbol{v}), \overline{x}, \hat{\boldsymbol{c}}]$	:2
	OP[2.1.4; 2.1.5] : $\neg\hat{p}[!\overline{x}, \hat{\boldsymbol{c}}] \vee \hat{\alpha}[\hat{f}(\boldsymbol{v}), \overline{x}, \hat{\boldsymbol{c}}]$	:2
	OP[**1.1.4**; **2.1.5′**]: $\neg\hat{p}[!\overline{x}, \hat{\boldsymbol{c}}] \vee \hat{\alpha}[\hat{f}(\boldsymbol{v}), \overline{x}, \hat{c}, \hat{\boldsymbol{c}}]$	:2
	RR[2.1.6] : $\neg\hat{p}[!\overline{x}, \hat{\boldsymbol{c}}] \vee \hat{\alpha}[\hat{f}(\boldsymbol{v}), \overline{x}, \hat{\boldsymbol{c}}]$	:2
2.2	$\neg\hat{\boldsymbol{p}}[!\overline{\boldsymbol{x}}, \hat{\boldsymbol{c}}]^{\sharp} \vee \hat{\alpha}[\overline{x}, \hat{\boldsymbol{c}}]$	:*Sel*
2.2.1	$\neg\underline{\hat{\boldsymbol{p}}[!\overline{\boldsymbol{x}}, \hat{\boldsymbol{c}}]} \vee \hat{\alpha}[\overline{x}, \hat{\boldsymbol{c}}]$	:OR.2
	OR[1.1.1; 2.2.1]: $\hat{\alpha}[\hat{c}]$	:1
	OR[2.1.1; 2.2.1]: $\neg\hat{p}[!\overline{x}, \hat{\boldsymbol{c}}] \vee \hat{\alpha}[\hat{f}(\boldsymbol{v}), \overline{x}, \hat{\boldsymbol{c}}]$	:2

in which all constant symbols are smaller than non-constant symbols. Restriction (*ii*) is needed to avoid paramodulation inferences from oriented equations of form $\underline{c} \simeq f(\boldsymbol{v})$, as this may result in increase of the number of variables in clauses.

Restriction (*i*) is needed to obtain optimal complexity results. For this, we have developed a special strategy according to which we *split* a ground clause into *unit* literals whenever its eligible literal is a positive equality (case 1.0). Since the eligible literal must be maximal in the clause, by (*i*), we have that *all* literals in this clause are equalities or inequalities between constants. Unit equalities that are produced after splitting in case 1.0, can be used in Ordered Paramodulation inferences (case 1.1.4). However, we apply them in a more general way for *eager simplification*. That is, whenever an equation of form $c_1 \simeq c_2$ is obtained with $c_1 \succ c_2$, we replace *all* occurrences of constant c_1 in *every* clause, with constant c_2.

It turns out that the direct procedure gives us essentially the same complexity as the one with elimination of constants: for the details see Appendix C.2.2.

Implications for description logics

The guarded fragment for which we have presented decision procedures, became large enough to capture some expressive description logics.

As has been pointed out in subsection 3.3.3, the guarded fragment captures the relational translation of modal formulas, and $\mathcal{ALC}$-concepts (see $\mathcal{ALC}$-fragment of first-order logic (3.9) on p.66). This also holds if we allow *inverse roles*. Hence *satisfiability* of $\mathcal{ALCI}$-concepts can be decided through the guarded fragment without equality and constants.

It is also possible to express simple *role hierarchies* in the guarded fragment: $S \sqsubseteq T$ is translated to $\forall xy.[S(x,y) \rightarrow T(x,y)]$. Moreover, the *general inclusion axioms* $C_1 \sqsubseteq C_2$ can be expressed in the guarded fragment *with equality*[5] by $\forall x.[(x \simeq x) \rightarrow (\tau(C_1,x) \rightarrow \tau(C_2,x))]$ (recall that both translations $\tau(C_1,x)$ and $\tau(C_2,x)$ are guarded formulas). So now the concept subsumption problem w.r.t. general TBoxes can be decided through the guarded fragment.

Finally, as has been already noted in the beginning of this section, constants in guarded fragment allow us to express *nominals*. Hence, the following result is proven in this section:

Corollary 4.3.7. *There is a paramodulation-based procedure which decides the concept subsumption problem w.r.t. general $\mathcal{ALCIOH}$-TBoxes in EXPTIME.*

4.3.3 Guarded Fragment and Functionality

Another class of constructors that are commonly used in description logics, but cannot be directly expressed in the guarded fragment, are *(qualified) number restrictions* and *functional restrictions*. Unfortunately, the guarded fragment becomes *undecidable* already with functional restrictions. In this section we define a *decidable* extension of the guarded fragment that captures description logics with functionality. We extend our paramodulation-based decision procedure to this fragment and show that the complexity of this extension remains the same both for its full and the bounded-variable versions.

Undecidability for the guarded fragment with functionality

We say that a binary relation $r(x,y)$ is *functional* if it satisfies the following axiom:

$$\mathtt{F.}\quad r(x,y)\ \wedge\ r(x,z) \rightarrow y \simeq z \qquad \textit{(functionality)}$$

Grädel [1999] has investigated a *guarded fragment with functionality*, which we denote by $\mathcal{GF}[\mathsf{functional}(\mathsf{r}_1,..,\mathsf{r}_i,..)]$, or, short $\mathcal{GF}[\mathsf{F}]$. $\mathcal{GF}[\mathsf{F}]$ is a set of guarded formulas, where some binary atoms are declared to be functional, i.e., the set of possible interpretations is restricted to those where the listed predicate symbols are interpreted

[5] although, equality is used here only in guards, and hence, can be completely eliminated

by functional relations. It turns out that satisfiability problem for $\mathcal{GF}[\mathsf{F}]$ is in general undecidable. In fact, this is already the case for $\mathcal{GF}^3[\mathsf{F}]$, i.e., for three-variable guarded fragment with one functional relation:

Theorem 4.3.8 ([Grädel, 1999])**.** $\mathcal{GF}^3[\mathsf{functional(r)}]$ *is a conservative reduction class, i.e., it is undecidable for (finite) satisfiability.*

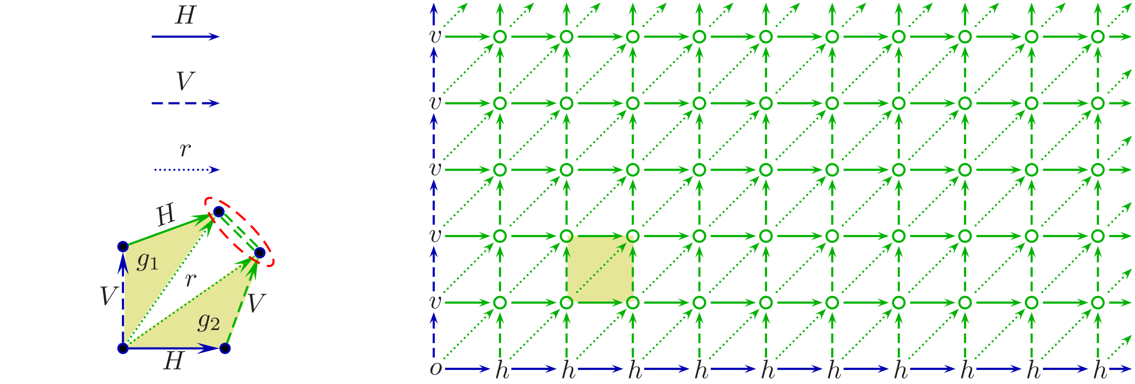
Proof. The proof of this theorem is by a reduction from domino problems (see section 3.4). Given a domino system $\mathcal{D} = (D, H, V)$ we construct a formula $F \in \mathcal{GF}^3[\mathsf{functional(r)}]$ such that F is (finitly) satisfiable *iff* $\mathcal{D}$ admits (periodic) tiling of a grid $\mathbb{N} \times \mathbb{N}$. Our formula F is a conjunction of formula TILING given in Figure 3.1, that encodes tiling conditions for a domino problem, and a formula GRID defined below:

$$\begin{array}{rcll}
\mathsf{GRID} & := & \exists x.[o(x) \wedge v(x) \wedge h(x)] \wedge & \text{- } \textit{constructs the origin of a grid} \\
& & \wedge\ \forall x.(v(x) \rightarrow \exists y.[V(x,y) \wedge v(y)]) \wedge & \text{- } \textit{launches the initial vertical axis} \\
& & \wedge\ \forall x.(h(x) \rightarrow \exists y.[H(x,y) \wedge h(y)]) \wedge & \text{- } \textit{launches the initial horizontal axis} \\
& & \wedge\ \forall xy.[V(x,y) \rightarrow \exists z.g_1(x,y,z)] \wedge & \text{- } \textit{creates the upper triangle of a cell} \\
& & \wedge\ \forall xy.[H(x,y) \rightarrow \exists z.g_2(x,y,z)] \wedge & \text{- } \textit{creates the right triangle of a cell} \\
& & \wedge\ \forall xyz.[g_1(x,y,z) \rightarrow H(y,z) \wedge r(x,z)] \wedge & \text{- } \textit{creates the remaining} \\
& & \wedge\ \forall xyz.[g_2(x,y,z) \rightarrow V(y,z) \wedge r(x,z)] & \text{- } \qquad\qquad \textit{edges of triangles}
\end{array} \tag{4.27}$$

Recall, that in order to enforce a grid structure, we need to encode a confluence property for relations V and H: $H^{\smile} \circ V \subseteq V \circ H^{\smile}$. This is done by the last four conjuncts from (4.27) using a condition that $r(x,y)$ must be a functional relation. Using these conjunctions we launch two triangles from incident vertical and horisontal edges of a cell and then "glue" them on diagonal into a cell using functionality of $r(x,y)$: see Figure 4.2. It is easy to show that every (periodic) tiling of an infinite quadrant $\mathbb{N} \times \mathbb{N}$ of a plane yields a (finite) model for $F = \mathsf{TILING} \wedge \mathsf{GRID}$ where $r(x,y)$ is functional, and vice versa, every (finite) model of F, where $r(x,y)$ is functional, can be unfolded into a (periodic) tiling of a grid. Hence this reduction proves that $\mathcal{GF}^3[\mathsf{functional(r)}]$ is a conservative reduction class. $\Box$

As has been pointed out in [Grädel, 1999], the result of Theorem 4.3.8 is optimal w.r.t. the number of variables used in the fragment, since already the two-variable fragment with counting $\mathcal{C}^2$, which subsumes $\mathcal{GF}^2[\mathsf{F}]$, is decidable [Grädel, Otto & Rosen, 1997; Pacholski et al., 2000].

Figure 4.2 Undecidability of $\mathcal{GF}^3$[functional(r)]



The guarded fragment with functional guards

Let us try to analyse and explain why the guarded fragment with functionality becomes undecidable, whereas this is not the case for many description logics with functionality restrictions. One explanation could be in that these logic usually correspond to *two-variable* guarded fragment with functionality, which is decidable. While it is possible to come up with a decision procedure for this fragment based on paramodulation (we return to this point later), we have found another explanation of this phenomenon.

If we look carefully at formula (4.27), we notice that the functional atom $r(x, y)$ is used here in the *body* of the guarded formula *positively*, which makes it possible to enforce functionality restrictions for some *larger* atoms: in this formula we enforce functionality of predicates $g_1(x, y, z)$ and $g_2(x, y, z)$ w.r.t. their first and the last arguments. This made it possible, in the end, to "glue" triangles corresponding to these atoms on one edge, while keeping the remaining vertices free. However, this does not happen in first-order formulas that correspond to description logics, not only because there are no relations of higher arity, but also because the functional roles, as any other roles, are typically used *in guards*.

Motivated by this observation, we consider the *guarded fragment with functional guards* $\mathcal{GF}$[functional_guards(r$_1$,..)] (short $\mathcal{GF}$[FG]), to be a set of formulas from $\mathcal{GF}$[functional(r$_1$,..)] in which functional predicate symbols r_1,.. occur in *guard positions only*. It can be shown that fragment $\mathcal{GF}$[FG], can in particular capture first-order translation for DL $\mathcal{ALCIF}$. Later we discuss how to extend this fragment in order to capture more description logics.

Remark 4.3.9. In the definition of guarded formulas (3.17) that we gave in subsection 3.3.3, it is allowed, in principle, to use the constructor $\forall\overline{y}.[G \rightarrow F_1]$ with an *empty* variable-vector $\overline{y}$ (since this will be just implication). However, when we restrict occurrences of certain predicate symbols to guards only, we assume that the usage of these guards is *non-trivial*, i.e., $\overline{y}$ is *non-empty*. Otherwise these restrictions make no sense, since these predicate symbols could appear essentially everywhere using quantification over the empty variable-vector. ◈

Let us see how functional guards from $\mathcal{GF}[\mathsf{FG}]$ may appear in clauses that result from **CNF**-translation of guarded formulas given in subsection 4.1.1. If we look at the clause types obtained in Table 4.1 after the structural transformation, we notice that guard atoms appear only in clauses of types (6) − (8). Now, by restricting guards in these clauses to *binary* functional predicate symbols, we obtain clause types listed in Table 4.18. We intentionally changed clausification in the last two cases in order to simplify the upcoming proofs, by introducing auxiliary unary definitional predicate symbols $\mathtt{p}'_F$. However, our procedure can be easily repeated when a direct Skolemization is applied. Functional restrictions can be encoded in first-order logic directly, using an axiom $\forall xyz.[r(x,y) \wedge r(x,z) \rightarrow x \simeq z]$. But it is easier to deal with the axiom $\forall xy.[r(x,y) \rightarrow y \simeq h'(x)]$, where $h'(x)$ is a fresh unary functional symbol introduced for atom $r(x,y)$.

Table 4.18 Clause types for guarded formulas with functional guards

Type of a conjunct	$\dashrightarrow [\cdot]^{sk},\ [\cdot]^{cnf} \dashrightarrow$	**Type of a clause**	(**Nr**)
$\forall x.(\mathtt{p}_F(x) \rightarrow \forall y.(r[!x,!y] \rightarrow \mathtt{p}_{F_1}[x,y]))$	$\dashrightarrow$	$\neg r[!x,!y] \vee \neg\mathtt{p}_F(x) \vee \mathtt{p}_{F_1}[x,y]$	(6.1)
$\mathtt{p}_F \rightarrow \forall xy.(r[!x,!y] \rightarrow \mathtt{p}_{F_1}[x,y])$	$\dashrightarrow$	$\neg r[!x,!y] \vee \neg\mathtt{p}_F \vee \mathtt{p}_{F_1}[x,y]$	(6.2)
$\mathtt{p}_F \rightarrow \forall x.(r(x,x) \rightarrow \mathtt{p}_{F_1}[x])$	$\dashrightarrow$	$\neg r(x,x) \vee \neg\mathtt{p}_F \vee \mathtt{p}_{F_1}[x]$	(6.3)
$\forall x.(\mathtt{p}_F(x) \rightarrow \exists y.(r[!x,!y] \wedge \mathtt{p}_{F_1}[x,y]))$	$\dashrightarrow$	$\neg\mathtt{p}_F(x) \vee r[!x,!h(x)]$	(7.1)
		$\neg\mathtt{p}_F(x) \vee \mathtt{p}_{F_1}[x,h(x)]$	(8.1)
$\mathtt{p}_F \rightarrow \exists xy.(r[!x,!y] \wedge \mathtt{p}_{F_1}[x,y]))$	$\dashrightarrow$	$\neg\mathtt{p}_F \vee \mathtt{p}'_F(\mathtt{c}_{sk})$	(7.2)
		$\neg\mathtt{p}'_F(x) \vee r[!x,!h(x)]$	(7.1)
		$\neg\mathtt{p}'_F(x) \vee \mathtt{p}_{F_1}[x,h(x)]$	(8.1)
$\mathtt{p}_F \rightarrow \exists x.(r(x,x) \wedge \mathtt{p}_{F_1}[x]))$	$\dashrightarrow$	$\neg\mathtt{p}_F \vee \mathtt{p}'_F(\mathtt{c}_{sk})$	(7.2)
		$\neg\mathtt{p}'_F(x) \vee r(x,x)$	(7.3)
		$\neg\mathtt{p}'_F(x) \vee \mathtt{p}_{F_1}[x]$	(8.2)

After that, we end up with clauses which we generalise to clause class ($\mathbf{G}^f$) defined in Table 4.19. The clauses of this class are either the usual guarded clauses 1 – 2, which may contain functional predicate symbols $r(x,y)$ only *negatively*, or clauses `F` and `U`.1 – `U`.4 containing functions $h(x)$ introduced for functional atoms $r(x,y)$ either after Skolemization, or from the functionality axioms. The important property of our saturation strategy is that we will keep these types of clauses separately, so they do not mix with each other, in a similar fashion as it has been done for combinations of fragments in section 4.2.

Table 4.19 A clause class for the guarded fragment with functional guards

	Clause scheme	*Description*
$(\mathbf{G}^f)$:	1 $\hat{\alpha}[\hat{c}]$	*a ground clause without positive occurrences of functional predicate symbols;*
	2 $\neg\hat{a}[!\overline{x}] \vee \hat{\alpha}[\hat{f}(\overline{x}), \overline{x}]$	*a guarded clause whose functional predicate symbols occur only negatively.*
	`F` $\neg r(x, y) \vee y \simeq h(x)$	*a clause expressing functionality of a binary relation, where $h(x)$ does not occur in clauses of form 2;*
	`U`.1 $\hat{\alpha}[h(x), x] \vee r[h(x), x]$	*a disjunction of a clause of form* `U`.4 *and an atom $r[h(x), x]$, where r is a functional predicate symbol and $h(x)$ is the same as in* `U`.4*;*
	`U`.2 $\hat{\alpha}[h(x), x] \vee h(x) \simeq h_1(x)$	*a disjunction of a clause of form* `U`.4 *and an atom $h(x) \simeq h_1(x)$, where $h_1(x)$ does not occur in clauses of form 2 either;*
	`U`.3 $\hat{\alpha}[h(x), x] \vee h_1(h(x)) \simeq x$	*the same as in the previous case but the atom is now of form $h_1(h(x)) \simeq x$.*
	`U`.4 $\hat{\alpha}[h(x), x]$	*a clause containing one variable and one unary functional term which does not occur in clauses of type 2*
	where $a := p \mid \simeq \mid r;\quad l := p \mid \neg p \mid \simeq \mid \not\simeq \mid \neg r;\quad \alpha := \vee\{l\}.$	

Saturation of the clause set

One of the problems that arise when applying paramodulation rules to clauses from $(\mathbf{G}^f)$, is that the *functional depth* of clauses *might grow.* Clauses with nested functional terms can already appear when a clause $\neg \mathsf{p}_{F_1}(x) \vee \underline{\boldsymbol{r(h_2(x), x)}}^{\star}$ of form (7.1), is resolved with a clause expressing functionality for r: $\underline{\boldsymbol{\neg r(x, y)}} \vee y \simeq h'(x)$, which results in a clause $\neg \mathsf{p}_{F_1}(x) \vee h'(h_2(x)) \simeq x$ of form `U`.3. However even deeper clauses can be obtained from clauses `U`.1 – `U`.4.

Example 4.3.10. Consider the conclusion of resolution inferences from two clauses of form `U`.2:

$$\underline{\boldsymbol{a(h_2(x))}}^{\star} \vee h_2(x) \simeq h_1(x) \qquad \textit{and} \qquad \neg\underline{\boldsymbol{a(h_2(x))}} \vee h_2(x) \simeq h_3(x) \tag{4.28}$$

The conclusion $\boldsymbol{h_2(x)} \simeq \underline{\boldsymbol{h_1(x)}}^{\star} \vee h_2(x) \simeq h_3(x)$ of this inference is already outside of $(\mathbf{G}^f)$. However, if we ignore this and paramodulate a clause $\underline{\boldsymbol{h_1(h_2(x))}} \simeq \boldsymbol{x}^{\star}$ of form `U`.3 to this clause, we obtain a clause $h_2(h_2(x)) \simeq x \vee \underline{\boldsymbol{h_2(h_2(x))}} \simeq \boldsymbol{h_3(h_2(x))}^{\star}$. The last clause is dangerous not because it contains nested functional terms, but because paramodulation from this clause into a subterm $h_2(x)$ of *any* clause would result in *increase of functional depth.* In particular, this clause can be paramodulated *into itself* which results in clauses of arbitrary depth (for simplicity we omit the remaining literal):

$$\begin{array}{lll}
1.\ \underline{\boldsymbol{h_2(h_2(x))}} \simeq \boldsymbol{h_3(h_2(x))}^{\star} & & :\texttt{OP.1} \\
2\ \ \boldsymbol{h_2(}\underline{\boldsymbol{h_2(x)}}\boldsymbol{)} \simeq \boldsymbol{h_3(}\underline{\boldsymbol{h_2(x)}}\boldsymbol{)} & & :\texttt{OP.2} \\
\texttt{OP}[1;2]\!:3 & \boldsymbol{h_2(h_3(}\underline{\boldsymbol{h_2(x)}}\boldsymbol{))} \simeq \boldsymbol{h_3(h_3(}\underline{\boldsymbol{h_2(x)}}\boldsymbol{))} & :\texttt{OP.2} \\
\texttt{OP}[1;3]\!:4 & \boldsymbol{h_2(h_3(h_3(}\underline{\boldsymbol{h_2(x)}}\boldsymbol{)))} \simeq \boldsymbol{h_3(h_3(h_3(}\underline{\boldsymbol{h_2(x)}}\boldsymbol{)))} & :\texttt{OP.2} \\
\quad \ldots\ldots\ldots \text{ etc.} & &
\end{array}$$

⟐

A similar problem has been dealt with in a recent paper [Hustadt et al., 2004], which studies decidability of some description logics including DL $\mathcal{SHIQ}$ by superposition. In order to avoid growth of term depth in clauses, the authors propose to use *basic strategies* [Bachmair, Ganzinger, Lynch & Snyder, 1995] which allow one to block inferences into terms that appear in variable positions after applying substitutions. For the particular example above, no paramodulation inference except for the first one, is necessary, since all subterms $h_2(x)$ occur in *substitutional positions*. However, even *basicness* does not prevent from nested terms in some cases, for which a special *decomposition rule* is introduced that decomposes deep clauses into shallow ones.

Here we propose a similar in spirit, but an easier (in our opinion) solution, based on the "*divide-and-conquer*" principle. The idea is, whenever a potentially dangerous equational literal of form $h_2(x) \simeq h_1(x)$ appears in a clause from $\texttt{U}.2$, this literal is "cut away" using the Literal Projection rule to a clause where we can afford certain growth in variable depth. It turns out that *no* supplementary *basic restrictions* are required in the end, to control the depth of clauses.

Continuing Example 4.3.10, according to our strategy, we do not resolve clauses from (4.28), but first apply the Literal Projection rule for their equational literals. These rule will simplify each of these clauses on two clauses:

$$\begin{array}{ccc}
\underline{\boldsymbol{a(h_2(x))}}^{\star} \vee p_{hh_1}(x) & & \neg\underline{\boldsymbol{a(h_2(x))}} \vee p_{hh_2}(x) \\
\neg p_{hh_1}(x) \vee \boldsymbol{h_2(x)} \simeq \underline{\boldsymbol{h_1(x)}}^{\star} & \textit{and} & \neg p_{hh_2}(x) \vee \underline{\boldsymbol{h_2(x)}} \simeq \boldsymbol{h_3(x)}^{\star}
\end{array} \tag{4.29}$$

Resolution between the first clauses gives us a clause $p_{hh_1}(x) \vee p_{hh_2}(x)$. After paramodulation of $\underline{\boldsymbol{h_1(h_2(x))}} \simeq \boldsymbol{x}^{\star}$ into the lower left clause, we obtain a clause $\neg p_{hh_1}(\underline{h_2(x)}) \vee \boldsymbol{h_2(}\underline{\boldsymbol{h_2(x)}}\boldsymbol{)} \simeq \boldsymbol{x}^{\star}$ of type $\texttt{U}.3$. Although this clause contains nested functional terms, they are not dangerous anymore, since terms in the equation have *different* variable depths. In particular, if we paramodulate this clause into the lower right clause we obtain: $\neg p_{hh_1}(h_2(x)) \vee \neg p_{hh_2}(h_2(x)) \vee x \simeq h_3(h_2(x))$, which is again of type $\texttt{U}.3$. If we paramodulate the previous clause into itself, we obtain a clause even of a *smaller* depth: $\neg p_{hh_1}(h_2(x)) \vee \neg p_{hh_1}(x) \vee h_2(x) \simeq x$.

In Table 4.20 we have listed all inferences between clauses from $(\mathbf{G}^f)$ that are possible according to our strategy. The inferences between clauses from 1 and 2

Table 4.20 Possible inferences between clauses for the guarded fragment with functional guards

$a := p \,|\, \simeq \,|\, r; \quad l := p \,|\, \neg p \,|\, \simeq \,|\, \not\simeq \,|\, \neg r; \quad \alpha := \vee\{l\}.$

1	$\hat{\alpha}[\hat{c}]$	:The cases are analogous to the those in Table 4.13
2	$\neg\hat{\boldsymbol{a}}[!\overline{x}] \vee \hat{\alpha}[\hat{f}(\overline{x}),\overline{x}]$	
2.1	$\neg\hat{\boldsymbol{a}}[!\overline{x}] \vee \hat{\alpha}[\hat{f}(\overline{x}),\overline{x}] \vee \boldsymbol{l}[!\hat{\boldsymbol{f}}(\overline{\boldsymbol{x}}),\overline{\boldsymbol{x}}]$	
2.1.1	$\neg\hat{\boldsymbol{a}}[!\overline{x}] \vee \hat{\alpha}[\hat{f}(\overline{x}),\overline{x}] \vee \underline{\boldsymbol{p}[!\hat{\boldsymbol{f}}(\overline{\boldsymbol{x}}),\overline{\boldsymbol{x}}]}^{\star}$	:OR.1
2.1.2	$\neg\hat{\boldsymbol{a}}[!\overline{x}] \vee \hat{\alpha}[\hat{f}(\overline{x}),\overline{x}] \vee \neg\underline{\boldsymbol{p}[!\hat{\boldsymbol{f}}(\overline{\boldsymbol{x}}),\overline{\boldsymbol{x}}]}$	:OR.2
2.1.2′	$\neg\hat{\boldsymbol{a}}[!\overline{x}] \vee \hat{\alpha}[\hat{f}(\overline{x}),\overline{x}] \vee \neg\underline{\boldsymbol{r}[!\hat{\boldsymbol{f}}(\overline{\boldsymbol{x}}),\overline{\boldsymbol{x}}]}$	:OR.2
2.1.3	$\neg\hat{\boldsymbol{a}}[!\overline{x}] \vee \hat{\alpha}[\hat{f}(\overline{x}),\overline{x}] \vee \underline{p[\hat{f}(\overline{x}),\overline{x}]} \vee \underline{\boldsymbol{p}[!\hat{\boldsymbol{f}}(\overline{\boldsymbol{x}}),\overline{\boldsymbol{x}}]}$	:OF
2.1.4	$\neg\hat{\boldsymbol{a}}[!\overline{x}] \vee \hat{\alpha}[\hat{f}(\overline{x}),\overline{x}] \vee \underline{\boldsymbol{f}(\overline{\boldsymbol{x}})} \simeq \{\overline{\boldsymbol{x}}\}^{\star}$	:OP.1
2.1.5	$\neg\hat{\boldsymbol{a}}[!\overline{x}] \vee \hat{\alpha}[\underline{f(\overline{x})},\hat{f}(\overline{x}),\overline{x}]$	:OP.2
2.1.6	$\neg\hat{\boldsymbol{a}}[!\overline{x}] \vee \hat{\alpha}[\hat{f}(\overline{x}),\overline{x}] \vee \underline{\boldsymbol{f}(\overline{\boldsymbol{x}})} \not\simeq \underline{\boldsymbol{f}(\overline{\boldsymbol{x}})}$	:RR
	OR[2.1.1; 2.1.2] : 2; OF[2.1.3] : 2; OP[2.1.4; 2.1.5] : 2; RR[2.1.6] : 2;	
2.2	$\neg\hat{\boldsymbol{a}}[!\overline{\boldsymbol{x}}]^{\sharp} \vee \hat{\boldsymbol{\alpha}}[\overline{\boldsymbol{x}}]$	:*Sel*
2.2.1	$\neg\underline{\hat{\boldsymbol{a}}[!\overline{\boldsymbol{x}}]} \vee \hat{\boldsymbol{\alpha}}[\overline{\boldsymbol{x}}]$	:OR.2
2.2.2	$\underline{\boldsymbol{x_1}} \not\simeq \underline{\boldsymbol{x_2}} \vee \hat{\boldsymbol{\alpha}}[\boldsymbol{x_1},\boldsymbol{x_2}]$	:RR
	OR[1.1.1; 2.2.1] : 1; OR[2.1.1; 2.2.1] : 2; RR[2.2.2] : U.4;	
F	$\neg\underline{\boldsymbol{r}(\boldsymbol{x},\boldsymbol{y})} \vee y \simeq h(x)$	:OR.2
U.**1**	$\hat{\alpha}[h(x),x] \vee r[h(x),x]$	
U.**1.1**	$[\![\,\hat{\alpha}[h(x),x] \vee \boldsymbol{r}[\boldsymbol{h}(\boldsymbol{x}),\boldsymbol{x}]^{\sharp}]\!]$	:LP
U.**1.3**	$\hat{\alpha}^{1}[x] \vee \boldsymbol{r}[\underline{\boldsymbol{h}(\boldsymbol{x})},\boldsymbol{x}]$	:OP.2
U.**1.2**	$\hat{\alpha}^{1}[x] \vee \underline{\boldsymbol{r}[\boldsymbol{h}(\boldsymbol{x}),\boldsymbol{x}]}^{\star}$	:OR.1
	LP[U.**1.1**] : U.**1.2**, U.**4**; OR[U.**1.2**; **2.2.1**] : U.**4**; OR[U.**1.2**; F] : U.**2**, U.**3**;	
U.**2**	$\hat{\alpha}[h(x),x] \vee h(x) \simeq h_1(x)$	
U.**2.1**	$[\![\,\hat{\alpha}[h(x),x] \vee \boldsymbol{h}(\boldsymbol{x}) \simeq \boldsymbol{h_1}(\boldsymbol{x})^{\sharp}]\!]$	:LP
U.**2.2**	$\hat{\alpha}^{1}[x] \vee \underline{\boldsymbol{h}(\boldsymbol{x})} \simeq \boldsymbol{h_1}(\boldsymbol{x})^{\star}$	:OP
	LP[U.2.1] : U.2.2, U.4; OP[U.2.2; U.1.3] : U.1; OP[U.2.2; U.2.2] : U.2.2;	
U.**3**	$\hat{\alpha}[h(x),x] \vee \boldsymbol{h_1}(\boldsymbol{h}(\boldsymbol{x})) \simeq \boldsymbol{x}$	
U.**3.1**	$\hat{\alpha}[h(x),x] \vee \underline{\boldsymbol{h_1}(\boldsymbol{h}(\boldsymbol{x}))} \simeq \boldsymbol{x}^{\star}$	:OP
U.**3.2**	$\hat{\alpha}[\underline{h(x)},x] \vee \boldsymbol{h_1}(\underline{\boldsymbol{h}(\boldsymbol{x})}) \simeq \boldsymbol{x}$	:OP.2
	OP[U.3.1; U.1.3] : U.1; OP[U.3.1; U.2.2] : U.3; OP[U.3.1; U.3.1] : U.3; OP[U.2.2; U.3.2] : U.3; OP[U.3.1; U.3.2] : U.2	
U.4	$\hat{\alpha}[h(x),x]$	
U.**4.1**	$\hat{\alpha}[h(x),x] \vee \underline{\boldsymbol{l}[!\boldsymbol{h}(\boldsymbol{x}),\boldsymbol{x}]}^{\star}$	:OR
U.**4.5**	$\hat{\alpha}[x] \vee \underline{\boldsymbol{l}[!\boldsymbol{x}]}^{\star}$	:OR
U.**4.2**	$\hat{\alpha}[x] \vee \underline{\boldsymbol{h}(\boldsymbol{x})} \simeq \boldsymbol{x}^{\star}$	:OP.1
U.**4.6**	$\hat{\alpha}[x] \vee \underline{\boldsymbol{x}} \not\simeq \underline{\boldsymbol{x}}$	:RR
U.**4.3**	$\hat{\alpha}[\underline{h(x)},x]$	:OP.2
U.**4.7**	$\hat{\alpha}[]$	⇒ :1
U.**4.4**	$\hat{\alpha}[h(x),x] \vee \underline{\boldsymbol{h}(\boldsymbol{x})} \not\simeq \underline{\boldsymbol{h}(\boldsymbol{x})}$	:RR

	OR[U.**4.1**; **2.2.1**] : U.4; OR[U.**4.1**; U.**4.1**] : U.4; OP[U.**4.2**; U.**1.3**] : U.**1**;
	OP[U.**4.2**; U.**2.2**] : U.4; [U.**4.2**; U.**3.2**] : U.4; OP[U.**4.2**; U.**4.3**] : U.4;
	OP[U.**2.2**; U.**4.3**] : U.4; OP[U.**3.1**; U.**4.3**] : U.4; RR[U.**4.4**] : U.4;
	OR[U.**4.5**; **1**.∗] : **1**; OR[U.**4.5**; **2.1**.∗] : **2**; OR[U.**4.5**; **2.2**.1] : U.4;
	OR[U.**4.5**; U.**4.5**] : U.4; RR[U.**4.6**] : U.4;	

are identical to those for ($\mathbf{G}^{\simeq}$) given in Table 4.13. The only difference is that instead of clauses of form U, we have now clauses of form U.4. As seen from this table, we apply the Literal Projection rule twice: in case U.2.1 and in case U.3.1. The preconditions for both applications of this rule is that the remaining part of the clause must contain either term $h_2(x)$, or a predicate symbol with arity greater than 1 (in this case it is a simplification rule). Note that applications of these rule introduce only finitely many new predicate symbols, since only equational or binary atoms where "projected".

The reasons for application of the Literal Projection rule for case U.3.1 where given in Example 4.3.10 above. For the case U.2.1 this rule is not strictly necessary, but we have applied it in order to simplify the case analysis (and keep things separate). The remaining inferences given in the table, are relatively straightforward.

The case analysis presented in Table 4.20 demonstrates that the set of clauses ($\mathbf{G}^{\simeq}$) is closed under our paramodulation strategy. By carrying out similar computations as in the case with $\mathcal{GF}_{\simeq}$, we obtain the following result:

Theorem 4.3.11. *There is a paramodulation-based decision procedure for the guarded fragment with equality and functional guards* $\mathcal{GF}_{\simeq}[\mathsf{FG}]$ *which can be implemented in 2EXPTIME. This procedure decides the bounded-variable version of this fragment* $\mathcal{GF}^k_{\simeq}[\mathsf{FG}]$ *in EXPTIME.*

Note that it is possible to admit *negative* occurrences of functional literals not only in guard positions, since this would result in the same clause class ($\mathbf{G}^f$).

On functional relations of greater arity

The fragments that we have considered above, have been restricted only to *binary* functional relations. It is a natural question, *whether our results can be carried out to arbitrary functional relations*? Unfortunatly an extension of the guarded fragment with functional guards of greater arity is undecidable. Indeed, using a ternary relation $r(x, y, z)$, which is functional for z w.r.t. x (and possibly w.r.t. y), we can enforce functionality of any other binary relation $a(x, y)$ using a guarded formula in which $r(x, y, z)$ occurs in a guard position:

$$\forall xz.(a(x,z) \rightarrow \exists y.[r(x,y,z) \wedge x \simeq y]) \tag{4.30}$$

Since the atom $a(x, y)$ is no longer restricted to occur in guards, we obtain undecidability of this fragment in a similar way as for the guarded fragment with functionality. An open question is then, *which additional conditions can be imposed on functional guards in order to retain decidability?* The only variants of the guarded fragment we are aware of, which admit for a certain form of counting for relations of greater arity, are so-called *action-guarded logics* introduced in [Goncalves & Grädel, 2000]

4.3.4 Guarded Fragment with Counting

Expressive description logics, that are considered nowadays, usually employ a generalisation of *functional restrictions*—so-called *(qualified) number restrictions* (see subsection 3.2.2). It would be rather useful to find a variant of the guarded fragment which allows to capture such constructors, and to extend our paramodulation-based procedure for them.

Although, as we will see in a moment, such extensions are possible and fairly straightforward, there are certain well-known complexity issues related to how the numbers are represented in formulas: in *unary coding* or in *binary coding*.

From functionality to counting

Before we describe an extension of the guarded fragment that captures (qualified) number restrictions in description logics, we consider a simple generalisation of functionality restrictions to *global number restrictions* of the forms:

$$\begin{array}{lll} \texttt{L.} & \forall x.\exists^{\geq n}y.r(x,y) & \text{(\textit{``at-least'' number restrictions)}} \\ \texttt{M.} & \forall x.\exists^{\leq m}y.r(x,y) & \text{(\textit{``at-most'' number restrictions})} \end{array} \tag{4.31}$$

where n and m are naturals. Informally, these restrictions express that for every element x there must be at least n, respectively at most m *different* elements y such that $r(x,y)$ holds. Formally, the semantics of these constructors is defined similarly as in Table 3.2 on p.67:

$$\begin{aligned} \forall x.\exists^{\geq n}y.r(x,y) &\equiv \forall x.\exists y_1..y_n.[\bigwedge_{1\leq i\leq n} r(x,y_i) \wedge \bigwedge_{1\leq i<j\leq n} (y_i \not\simeq y_j)] \\ \forall x.\exists^{\leq m}y.r(x,y) &\equiv \forall x.\forall y_1..y_{m+1}.[\bigwedge_{1\leq i\leq m+1} r(x,y_i) \rightarrow \bigvee_{1\leq i<j\leq m+1} (y_i \simeq y_j)] \end{aligned} \tag{4.32}$$

Quantifiers of form $\exists^{\geq n}y.$ and $\exists^{\leq m}y.$ are also called *counting quantifiers*. Note that functionality restrictions are instances of "at-most" number restrictions when $m = 1$.

Let us see which clauses correspond to number restrictions. By skolemizing the first formula above, we obtain the clauses:

$$\begin{array}{lll} \texttt{L.}1 & r(x,h_i(x)) & 1\leq i\leq n; \\ \texttt{L.}2 & h_i(x)\not\simeq h_j(x) & 1\leq i<j\leq n; \end{array} \tag{4.33}$$

where $h_i(x)$ is a Skolem function introduced for i-th variable. The second formula can be expressed similarly as functional restrictions, using m auxiliary *counting functions* $h^1(x),\ldots,h^m(x)$ introduced for a binary relation $r(x,y)$:

$$\texttt{M.}1 \quad \underline{\neg \boldsymbol{r(x,y)}}^{\sharp} \vee y\simeq h^1(x)\vee\cdots\vee y\simeq h^m(x) \tag{4.34}$$

We can notice that clauses of form L.1 are instances clause scheme F.2 from Table 4.19, and clauses M.1 and L.2 are close to schemes F.1 and F.3 respectively. In fact, these clauses can be treated in saturation essentially in the same way as those from $(\mathbf{G}^f)$. For example, clause of form M.1 can be resolved only with clauses of form F.2, after which the clause is *split* into clauses of form F.3 or F.4 using the Literal Projection rule.

In order to avoid clauses of form L.2, one can express "at-most" number restrictions directly in $(\mathbf{G}^f)$. This can be done by introducing n additional binary atoms $e_i(x, y)$ for i with $1 \le i \le n$, and replacing L.2 with the following clauses of form F.5:

$$\begin{array}{lll} \texttt{L.2.1} & e_i(x, h_i(x)) & 1 \le i \le n; \\ \texttt{L.2.2} & \neg e_i(x, h_j(x)) & 1 \le i < j \le n. \end{array} \tag{4.35}$$

However a direct procedure should be considered as more efficient for the practice.

Let us generalise this approach in order to capture (qualified) number restrictions in description logics. For this purpose, we introduce a *guarded fragment with number restrictions* $\mathcal{GFN}$:

$$\mathcal{GFN} \ ::= \ A \mid \neg F_1 \mid F_1 \wedge F_2 \mid \forall \overline{y}.[G \rightarrow F_1] \mid \boldsymbol{\exists^{\le n} y.[r(x, y) \wedge F_1]} \ . \tag{4.36}$$

The last is a new constructor for this fragment, for which we have ($\boldsymbol{i}$) *free*$[F_1] \subseteq \{x, y\}$ (in other words, $r(x, y)$ is a *counting guard*) and ($\boldsymbol{ii}$) atom $r(x, y)$ is restricted to occur only as a guard (i.e., it may not be used in the base case). The semantics for the *counting quantifiers* is defined analogous to (4.32). It is easy to see that (qualified) number restrictions can be expressed in $\mathcal{GFN}$: see Table 3.2 on p.67.

Let us see, which types of clauses correspond to $\mathcal{GFN}$. Note that the occurrence of formula F_1 in a formula $\exists^{\le n} y.[r(x, y) \wedge F_1]$ is *negative* (see the first-order translation for counting quantifiers above), hence the *negation normal form* for formulas in $\mathcal{GFN}$ is defined by:

$$[\mathcal{GFN}]^{nnf} \ ::= \ (\neg)A \mid F_1 \mathbin{\wedge\!\!\!\!\vee} F_2 \mid \ \cdots \ \mid \exists^{\le n} y.[r(x, y) \wedge \neg F_1] \mid \exists^{\ge n} y.[r(x, y) \wedge F_1] \ . \tag{4.37}$$

By applying the usual structural transformation and Skolemization we obtain new clause types for $\mathcal{GFN}$ listed in Table 4.21. We see that the clauses for $\mathcal{GFN}$ are very similar to clauses that we have obtained for *global number restrictions*. So, they can be dealt with in essentially the same way. For example, if we select literal $\neg r(x, y)$ in clause of form (N.1), it can be resolved only with clauses of form F.2 from Table 4.19, and the result can be split into clauses of form F.3 and F.4 using the Literal Projection rule. Treatment for clauses of other types is similar. Hence the following result is obtained analogously to $\mathcal{GF}$[FG]:

Table 4.21 Additional clause types for guarded formulas with number restrictions

Type of a conjunct	$\dashrightarrow [\cdot]^{sk},\ [\cdot]^{cnf} \dashrightarrow$	**Type of a clause**		**(Nr)**
$\forall x.(\mathtt{p}_F(x) \rightarrow \exists^{\leq n} y.[r(x,y) \wedge \neg \mathtt{p}_{F_1}[x,y]])$	$\dashrightarrow$	$\neg r(x,y) \vee \neg \mathtt{p}_F(x) \vee$ $\vee\ \mathtt{p}_{F_1}[x,y] \vee \bigvee_{1 \leq i \leq n} y \simeq h^i(x)$		(N.1)
$\forall x.(\mathtt{p}_F(x) \rightarrow \exists^{\geq n} y.[r(x,y) \wedge \mathtt{p}_{F_1}[x,y]])$	$\dashrightarrow$	$\neg \mathtt{p}_F(x) \vee r(x, h_i(x))$,	$1 \leq i \leq n$	(N.2)
		$\neg \mathtt{p}_F(x) \vee \mathtt{p}_{F_1}[x, h_i(x)]$,	$1 \leq i \leq n$	(N.3)
		$\neg \mathtt{p}_F(x) \vee h_i(x) \not\simeq h_j(x)$,	$1 \leq i < j \leq n$	(N.4)
Alternative clauses for clauses of form (N.4):		$\neg \mathtt{p}_F(x) \vee e_i(x, h_i(x))$,	$1 \leq i \leq n$	(N.4.1)
		$\neg \mathtt{p}_F(x) \vee \neg e_i(x, h_j(x))$,	$1 \leq i < j \leq n$	(N.4.2)

Theorem 4.3.12. *There is a paramodulation-based decision procedure for the guarded fragment with number restrictions $\mathcal{GFN}$ which can be implemented in 2EXPTIME. This procedure decides the bounded-variable version of this fragment $\mathcal{GF}^k\mathcal{N}$ in EXPTIME. All complexities assume the unary coding of number restrictions.*

On unary and binary codings of numbers

Let us comment on the last sentence of Theorem 4.3.12. When complexity of logics with counting is analysed, there are generally two ways to measure the size of the input formula depending on the *coding of number restrictions*. For *unary coding* one assumes that the size of a counting quantifier $\exists^{\leq n} y.$ is *linear* in n, i.e., one assumes that these quantifiers are written as $\exists^{\leq \underbrace{11\ldots11}_{n\ times}} y$. However, it is more efficient to write number restrictions using a *binary coding of numbers*. In this case the size of the quantifier $\exists^{\leq n} y.$ is *logarithmic* in n.

Unfortunatly the direct complexity computation for the paramodulation-based procedure described in the previous section, does not give us satisfactory complexity results for $\mathcal{GFN}$ w.r.t. *binary coding* of numbers, since we introduce too many functional and predicate symbols by our **CNF**-translation from Table 4.21—possibly *exponentially many* w.r.t. the size of a formula with *binary coding* of numbers. Hence, all results formulated in Theorem 4.3.12 are one exponent higher for this case.

It is possible to deal with number restrictions effectively using a *polynomial encoding* of number restrictions in first-order logic. In [Kazakov, 2004], we have presented a translation that eliminates counting quantifiers from guarded formulas with number restrictions. Inspired by an automata-based decision procedure for DL $\mathcal{ALCQI}$ [see Tobies, 2001, Chapter 4], we have proposed a translation from the two-variable guarded fragment with number restrictions $\mathcal{GF}^2\mathcal{N}$ into the three-variable guarded fragment $\mathcal{GF}^3$. This translation preserves (un)satisfiability of formulas, and, importantly, *it is polynomial even if the number restrictions are coded in binary*:

Theorem 4.3.13 ([Kazakov, 2004])**.** *For any formula $F \in \mathcal{GF}^2\mathcal{N}$ there exists a formula $F' \in \mathcal{GF}^3$ such that (**i**) F is satisfiable iff F' is satisfiable, (**ii**) $|F'| = O(|F|)$ and (**iii**) F' can be computed in polynomial time from F. All sizes assume binary coding of numbers.*

Unfortunately, we cannot discuss the details of this result here, since this will lead us outside the main topic of this thesis. We just comment, that the main idea behind the translation is a *tree-model property* for $\mathcal{GF}^2\mathcal{N}$, which makes it possible to encode models using guarded formulas without number restrictions.

Implications for description logics

Paramodulation-based decision procedures for extensions of the guarded fragment with functionality and number restrictions, can be used to perform reasoning in many expressible description logics. As has been noted, guarded fragment with functional guards can be used for reasoning in DL $\mathcal{ALCIF}$. Similarly, it is possible to show that DL $\mathcal{ALCQI}$ (an extensions of $\mathcal{ALCI}$ with *qualified number restrictions*) can be translated into the two-variable guarded fragment with number restrictions $\mathcal{GF}^2\mathcal{N}$. In fact, for fragment $\mathcal{GF}^2\mathcal{N}$ a restrictions (*ii*) for definition (4.36) requiring that counting atoms $r(x, y)$ must occur only as guards, can be dropped: a proof of Theorem 4.3.13 does not use it. Hence, it is possible to capture even more expressive description logics like $\mathcal{ALCQIH}b$, which is $\mathcal{ALCQI}$ extended with *role hierarchies* and restricted Boolean combinations of roles.

Let us briefly compare two algorithms for reasoning in $\mathcal{ALCQI}b$-TBoxes which are optimal for *binary coding* of numbers. The first one, presented in [Tobies, 2001] employs a translation of a concept-subsumption problem into a *looping tree automaton*, which is then checked for emptiness. The second procedure described in [Kazakov, 2004] employs a translation through the guarded fragment (first, concepts and TBox are translated to $\mathcal{GFN}$, and then counting quantifiers are eliminated): see Figure 4.3. Both procedures give theoretically optimal complexity. However the automata-based procedure from [Tobies, 2001], requires the construction of an *exponentially* large automaton in the *first* step, after which a linear time emptiness test is performed. In contrast to this, the main complexity of our procedure is concentrated in the *last step*, where a decision procedure for the guarded fragment is employed. This makes our procedure more attractive from the practical point of view, since any optimised decision procedure for the guarded fragment (say, those given in this thesis or in [Hladik, 2002]) can be directly reused for $\mathcal{ALCQI}b$.

We hope that despite its non-optimal behaviour for the binary coding of numbers, the paramodulation-based decision procedure for $\mathcal{GFN}$ given in this section, can be

Figure 4.3 The outline of decision procedures for $\mathcal{ALCQI}b$ and $\mathcal{GFN}$: the dashed arrows represent an automata-based approach; the solid arrows represent a decision procedure through the guarded fragment; translations indicated by double arrows are described in [Kazakov, 2004]

$$\begin{tikzcd}
 & \mathcal{ALCQI}b \arrow[r, Rightarrow, "\mathbf{PTIME}"'] \arrow[dl, dashed, "\mathrm{EXPTIME}"'] & \mathcal{GF}^2\mathcal{N} \arrow[dr, Rightarrow, "\mathbf{PTIME}"] & \\
\mathit{Automata} \arrow[dr, dashed, "\mathrm{PTIME}"'] & & & \mathcal{GF}^3 \arrow[dll, "\mathrm{EXPTIME}"] \\
 & \{\bot,\top\} & &
\end{tikzcd}$$

still used in practise, since numbers restrictions in real knowledge bases are typically not that big. A particular challenge is to combine this decision procedure with a decision procedure given in subsection 4.3.2. The resulting fragment—$\mathcal{GFN}$ *with constants*—can be used to reason in DL $\mathcal{ALCQIO}$, which is known to lack the *tree-model property* and is NEXPTIME-complete [Tobies, 2000]. Because of the first, no tableaux algorithm was known for this logic until recently [the first one has appeared in Horrocks & Sattler, 2005]. Saturation-based decision procedures do not rely on a tree-model property, so we think that extending our decision procedure to $\mathcal{GFN}$ with constants should not be that difficult.

4.4 Conclusions

In this chapter we have presented and compared a variety of saturation-based decision procedures for several fragments of first-order logics, their combinations and extensions. The main results of this chapter can be summarised as follows:

1. We derived resolution-based decision procedures for the *guarded*, *two-variable* and *full monadic* fragments without equality of the respective complexities: TIME($2^{\boldsymbol{n}\cdot 2^{\boldsymbol{w}\cdot(\log \boldsymbol{w}+\epsilon)}}$), NTIME($2^{O(\boldsymbol{n})}$), NTIME($2^{O(\boldsymbol{n})}$), where $\boldsymbol{n}$ and $\boldsymbol{w}$ are the size and the width of the input formula. These complexities are optimal and best known from the literature, to the best of our knowledge.

2. We introduced a notion of *structural combination* of recursively defined fragments of first-oder logic and demonstrated how decision procedures for combinations of the *guarded*, *two-variable* and *full monadic* fragments without equality, can be derived by combining the resolution decision procedures for their components. We also demonstrated that almost all decidability results for the combinations of the above fragments do not hold with equality: the fragments $\mathcal{GF}^3_{\simeq}|\mathcal{FO}^2$, $\mathcal{GF}^3|\mathcal{FO}^2_{\simeq}$ and $\mathcal{GF}^3|\mathcal{FO}^2|\mathcal{M}_{\simeq}$ are undecidable.

3. We revisited a paramodulation-based decision procedure for the *guarded fragment with equality* [Ganzinger & de Nivelle, 1999], and extended this procedure to capture description logics with *nominals* and *counting restrictions*. We showed that nominals can be expressed within the *guarded fragment with equality and constants*, and many DLs with number restrictions can be captured by the *guarded fragment with number restrictions* $\mathcal{GFN}$, where the counting relations are allowed to occur in guards only. For both of these fragments we derived optimal paramodulation-based decision procedures.

Let us briefly summarise our experience with designing saturation-based decision procedures:

Uniformness: In our presentation, all decision procedures are obtained in a *uniform way* by a sequence of *standard* transformations: a *negation normal form* transformation, followed by a *structural transformation*, followed by *Skolemization*. It turns out that this approach yields *small clause classes* and *sharp complexity bounds*. We have also observed, that a clause class for a fragment can be obtained almost immediately from a *recursive definition* for its formulas.

Modularity: Uniform description of decidable fragments and their decision procedures, opens new perspectives in obtaining more expressive decidable fragments by *combining* constructors used in their recursive definitions. We have seen that in many cases, a decision procedure for a combined fragment can be obtained by taking a *union* of the respective clause classes, and only *cross-inferences* between clauses for *different* fragments need to be inspected (see section 4.2).

Correctness: In contrast to tableau-based and related procedures for first-order fragments, correctness of saturation-based procedures does not rely on *global properties* of a fragment (like the finite model property, or the tree-model property), but on *refutational completeness* of standard calculi (resolution and paramodulation) which is proven once-and-for-all. What usually remains to be shown is *termination* of a chosen saturation strategy, which is easier to demonstrate formally.

Complexity analysis: Saturation-based decision procedures provide not only decidability proofs for the considered fragments, but also make it possible to extract *best known complexity* bounds. Although in some cases straightforward complexity calculations do not give fully satisfactory results, the decision procedures themselves are optimal. Refined calculations which use certain properties of saturation rules, reveal much better complexity bounds.

Expressiveness: Our decision procedures were shown to be expandable for new constructors and features of fragments. We have seen that decision procedures for extensions of the *guarded fragment* with *equality*, *constants*, *functionality* and *number restrictions*, are obtained by appropriate modifications of the clause class for the guarded fragment. For some extensions, even several such solutions are possible (see subsection 4.3.2). By specialisation of these decision procedures, one can obtain reasoning algorithms for many expressive *description logics*.

Simplification rules: Our decision procedures are typically not based on pure *ordered resolution* or *ordered paramodulation* calculi, but are often enhanced with optional simplification rules. Simplification rules have been proven to be extremely useful in automated reasoning, since they allow one to effectively prune the search space of a prover. For saturation-based decision procedures, additional simplification rules (like Elimination of Duplicate Literals and Splitting) are indispensable as well. We have also discovered that *dynamic renaming techniques* and in particular the Literal Projection rule, turned out to be very useful: this rule makes it possible to simplify some saturation procedures, which otherwise require certain *non-standard refinements* of theorem proving, like *non-liftable orders* or *basic strategies* (see subsection 4.1.2 and a discussion related to Example 4.3.10).

Translational methods: In some cases, when direct saturation-based procedures do not give acceptable complexity results (like for $\mathcal{GFN}$ with binary coding of numbers), it is possible to employ translations based on model-theoretic arguments, to fragments that can be decided optimally. However one should take such translations with a grain of salt, since they often destroy the structure of an input formula, which may spoil a positive effect of subsequent optimisations [see Horrocks et al., 2000, Section 7].

Clause schemes: There are certain issues that should be addressed in future works. Among them is the *scheme notation* which needs to mature. In this thesis we introduced *clause schemes* in order to describe saturation procedures in concise way. However, in broader perspectives, *clause schemes* can provide formal tools for a machine-based development of saturation-based procedures, and in this case one should come up with a more robust syntax. Although some features needed to be reflected in clause schemes can be already seen from our presentation, it is not yet clear which clause classes needed to be captured and what is an appropriate language to express them.

Chapter 5

Guarded Fragment over Compositional Theories

In this chapter we study extensions of the *guarded fragment* with theories that can be characterised by a set of *compositional axioms* of form $S \circ T \subseteq H$, where S, T and H are binary relations.

We have already seen in section 2.5 that such axioms are extremely difficult to handle, since they easily lead to undecidable logics. On the other hand, there is a big demand in extensions of modal languages and in particular description logics with compositional axioms. In the first section of this chapter we outline some examples of such applications: (1) classification of medical terminologies with complex role dependencies, (2) reasoning about geographical data, in particularly about distances, routes and topological relations, and (3) interval-based temporal reasoning.

Motivated by such applications, we consider extensions of the guarded fragment with compositional relations. Our objective is on the one hand, to identify dangerous usage of compositional axioms that might lead to undecidability, and on the other hand to design decision procedures for useful decidable fragments with compositional axioms and study their complexities.

We start our exploration from extensions of the guarded fragment with the simplest form of compositional axioms—*transitivity*: $T \circ T \subseteq T$. Despite its simplicity, inaccurate usage of transitivity axiom easily leads to undecidable logics. We sharpen existing undecidability results and demonstrate that *two-variable guarded fragment* with *two* transitive relations is already *undecidable*.

A series of works [Ganzinger et al., 1999; Szwast & Tendera, 2001; Kieronski, 2003] has been devoted to the question of finding suitable extensions of the guarded fragment with transitivity that can generalise modal logics over transitive frames. These studies have resulted in the *guarded fragment with transitive guards*. In order to obtain a saturation-based decision procedure for this fragment, we generalise the idea of de Nivelle [1999] and design a special inference rule Transitive Closure. This rule makes it possible to block certain dangerous inferences with transitive atoms. A novel feature here is that unlike other simplification rules, the Transitive Closure rule does not make clauses redundant, but *inferences*. Using this rule, we will formulate the first *practical* decision procedure for the guarded fragment with transitive guards without equality based on the *ordered chaining calculus*. This procedure is practical not only because it gives the optimal complexity for the full fragment, but also because it is optimal, without further modifications, for many *simpler* subfragments that correspond to modal logics.

A similarity between the treatment of *transitivity* axioms and *associative compositional axioms* in chaining calculi took us to a thought that our procedure could be extended to a larger class of compositional axioms. And indeed, we found a decision procedure for a guarded fragment over *associative* compositional axioms, where *conjunctions* of compositional relations can be used as guards. Such extensions can be used for expressing description logics over metric distances and interval temporal relations. We have also tried to generalize this result to *relational algebras*, given by a collection of *extended* compositional axioms of form $S \circ T \subseteq H_1 \vee \cdots \vee H_n$ which admit many regular properties (and in particular *associativity*). Unfortunately such extensions are *undecidable* in general, which we show by a reduction from domino problems.

Finally, we are concerned with extensions of our decidability results to the case with equality. It turns out that presence of equality makes most of our extensions *undecidable*. The only exception is the *guarded fragment with transitive guards* [Szwast & Tendera, 2001], for which it is possible to extend our saturation-based decision procedure.

5.1 Background

5.1.1 Examples and Applications of Compositional Theories

In this section we give an overview of some works and topics which are closely related to compositional theories.

Complex role inclusion axioms.

Formalisation of medical nomenclature is nowadays one of the major application arias for description logics. For designing anatomical ontologies, one often needs to *propagate* certain properties of objects over relations [Rector, 2002]: for example, a disease of a part-of a bone, is also a disease of a bone. Moreover, many anatomical relations, such as $\mathsf{part\text{-}of}$ or $\mathsf{located\text{-}in}$ admit certain non-trivial interaction properties [Schulz & Hahn, 2001], which makes it harder to reason about them, for example:

$$\mathsf{part\text{-}of} \circ \mathsf{located\text{-}in} \sqsubseteq \mathsf{located\text{-}in}$$

In order to support reasoning with such interaction axioms, Horrocks & Sattler [2004] have considered extensions of DL $\mathcal{SHIQ}$ with *complex role inclusions axioms* of forms:

$$(\mathbf{1})\ S \sqsubseteq T, \qquad (\mathbf{2})\ S \circ T \sqsubseteq T, \quad \textit{or} \quad (\mathbf{3})\ T \circ S \sqsubseteq T. \tag{5.1}$$

Such axioms generalise transitive roles in a non-trivial way, but are still more restricted than arbitrary compositional axioms of form $S \circ T \sqsubseteq H$.

Horrocks & Sattler [2004] have demonstrated, that $(\boldsymbol{i})$ an extension of $\mathcal{SHIQ}$ with this very restricted form of compositional axioms is still *undecidable*, but $(\boldsymbol{ii})$ an extensions of $\mathcal{SHIQ}$ with *acyclic* axioms of form (5.1) is *decidable*. Here *acyclic* means that the *dependency relation* $\mathrel{\sqsubseteq\!\!\!\!\circ}$ on roles, defined by $S \mathrel{\sqsubseteq\!\!\!\!\circ} T$ *iff* one of axioms (1) – (3) holds for $S \neq T$, is well-founded. For the case (ii), a tableaux algorithm deciding subsumption in the resulted description logic was given.

Regular grammar logics.

A positive result from Horrocks & Sattler [2004] could be probably explained using a more general framework. As has been mentioned in the end of subsection 3.2.1, there is a close relationship between compositional axioms of form $R \circ S \subseteq T$ and *production rules* for *context-free grammars* $T \to RS$. del Cerro & Panttonen [1988], Baldoni et al. [1998] and later Demri [2001], Demri & de Nivelle [2005] have used this relationship for characterising (un)decidable modal logics based on inclusion axioms, called *grammar logics*. Similar to other logics over composition axioms, *grammar logics* are undecidable in general. However, there is a quite large subclass of *decidable grammar logics*, namely *regular grammar logics*, whose *compositional axioms* $S \circ T \subseteq H$ correspond to production rules of *regular* grammars.

Regular grammar logics subsume many well-known decidable modal logics, in particular those based on *transitive* and *Euclidean* frames (see Table 3.1). It is possible to show that, in fact, *acyclic* role inclusion axioms of form (5.1) correspond to regular languages. That is, for every role T the set of all strings $L(T) = S_1 \ldots S_n$ such that $S_1 \circ \cdots \circ S_n \sqsubseteq T$, is *regular*. This can be demonstrated by induction on T w.r.t. ordering $\mathrel{\sqsubseteq\!\!\!\!\circ}$: $(\boldsymbol{i})$ if T is maximal w.r.t. $\mathrel{\sqsubseteq\!\!\!\!\circ}$, i.e., when $S \mathrel{\sqsubseteq\!\!\!\!\circ} T$ for no S, we may have only rule $T \circ T \sqsubseteq T$ that produces T, hence a language for T might be either empty, or $L(T) = T^+$; $(\boldsymbol{ii})$ in the induction step, a language for T can be constructed by taking $L_0(T) := (L(S_1) \cup \cdots \cup L(S_l))^* \cdot (L(S'_1) \cup \cdots \cup L(S'_m)) \cdot (L(S''_1) \cup \cdots \cup L(S''_n))^*$, where $S_i \circ T \sqsubseteq T$, $S'_j \sqsubseteq T$, $T \circ S''_k \sqsubseteq T$, $S_i \neq T$, $S'_j \neq T$, $S''_k \neq T$, for all i, j, and k

with $1 \leq i \leq l$, $1 \leq j \leq m$ and $1 \leq k \leq n$, and $L(T) := L_0(T)^+$, if $T \circ T \sqsubseteq T$ and then $L(T) := L_0(T)$ otherwise.[1]

It can be shown that grammars that correspond to *associative* compositional axioms (see subsection 3.5.4, p.83) are regular as well. Conversely, every regular grammar can be "completed" by introducing auxiliary symbols to fulfil the associativity property. In short, every regular language can be represented by a *left-linear* grammar, i.e., by the set of production rules of forms (i) $A \to aB$ or (ii) $C \to b$, where a and b are *terminals* [see Hopcroft & Ullman, 1979]. Now, for every production rule of the first form, we introduce a new symbol P_{AB} and add a new production rule $P_{AB} \to a$. We also add production rules $A \to P_{AB}\, B$ and $P_{AC} \to P_{AB}\, P_{BC}$ for every *non-terminals* A, B and C. It is easy to check that the resulted grammar generates the same language and corresponds to associative compositional axioms.

Recently Demri & de Nivelle [2005] described a translation from *regular grammar logics with converses* to $\mathcal{GF}^2$, providing thereby a simple method for deciding many non-trivial modal logics.

Reasoning about distances.

The original purpose for the *chaining calculus* introduced in [Bachmair & Ganzinger, 1998*b*, 1995] was to optimise saturation-based theorem provers for reasoning with different kinds of compositional relations, and in particular, orderings. We have already given an example (3.21) which express compositional axioms for a *quasi-ordering* $\succsim$. It turns out, that *theories of metric distances* have a very similar properties.

Consider a set of binary relations of forms $x\, \mathtt{D}_{\leq n}\, y$, and $x\, \mathtt{D}_{>n}\, y$, where n is a natural number, which we call the *distance relations*. These relations intuitively express that *"the distance between x and y is smaller or equal, respectively greater than n"*. It is easy to see that the usual triangle inequalities for distances give us the following compositional axioms for the *distance relations*:

$$\textbf{(1)}\ \mathtt{D}_{\leq n} \circ \mathtt{D}_{\leq m} \subseteq \mathtt{D}_{\leq(m+n)} \quad \textbf{(2)}\ \mathtt{D}_{>(m+n)} \circ \mathtt{D}_{\leq n} \subseteq \mathtt{D}_{>m} \quad \textbf{(3)}\ \mathtt{D}_{\leq n} \circ \mathtt{D}_{>(m+n)} \subseteq \mathtt{D}_{>m} \qquad (5.2)$$

for every natural m and n. It is easy to verify that these compositional axioms are associative. Indeed, there are only four possible ways to "overlap" these axioms:

[1] A similar construction has been used in the proofs from [Horrocks & Sattler, 2004], however the authors do not draw parallels with regular languages

1. $(\mathtt{D}_{\leq n_1} \circ \mathtt{D}_{\leq n_2}) \circ \mathtt{D}_{\leq n_3} = \mathtt{D}_{\leq (n_1+n_2)} \circ \mathtt{D}_{\leq n_3} = \mathtt{D}_{\leq (n_1+n_2+n_3)}$;
   $\mathtt{D}_{\leq n_1} \circ (\mathtt{D}_{\leq n_2} \circ \mathtt{D}_{\leq n_3}) = \mathtt{D}_{\leq n_1} \circ \mathtt{D}_{\leq (n_2+n_3)} = \mathtt{D}_{\leq (n_1+n_2+n_3)}$;
2. $(\mathtt{D}_{>(m+n_1+n_2)} \circ \mathtt{D}_{\leq n_1}) \circ \mathtt{D}_{\leq n_2} = \mathtt{D}_{>(m+n_2)} \circ \mathtt{D}_{\leq n_2} = \mathtt{D}_{>m}$;
   $\mathtt{D}_{>(m+n_1+n_2)} \circ (\mathtt{D}_{\leq n_1} \circ \mathtt{D}_{\leq n_2}) = \mathtt{D}_{>(m+n_1+n_2)} \circ \mathtt{D}_{\leq (n_1+n_2)} = \mathtt{D}_{>m}$;
3. $(\mathtt{D}_{\leq n_1} \circ \mathtt{D}_{\leq n_2}) \circ \mathtt{D}_{>(m+n_1+n_2)} = \mathtt{D}_{\leq (n_1+n_2)} \circ \mathtt{D}_{>(m+n_1+n_2)} = \mathtt{D}_{>m}$;
   $\mathtt{D}_{\leq n_1} \circ (\mathtt{D}_{\leq n_2} \circ \mathtt{D}_{>(m+n_1+n_2)}) = \mathtt{D}_{\leq n_1} \circ \mathtt{D}_{>(m+n_1)} = \mathtt{D}_{>m}$;
4. $(\mathtt{D}_{\leq n_1} \circ \mathtt{D}_{>(m+n_1+n_2)}) \circ \mathtt{D}_{\leq n_2} = \mathtt{D}_{>(m+n_2)} \circ \mathtt{D}_{\leq n_2} = \mathtt{D}_{>m}$;
   $\mathtt{D}_{\leq n_1} \circ (\mathtt{D}_{>(m+n_1+n_2)} \circ \mathtt{D}_{\leq n_2}) = \mathtt{D}_{\leq n_1} \circ \mathtt{D}_{>(m+n_1)} = \mathtt{D}_{>m}$.

Hence, it is possible to use the *ordered chaining calculus* $\mathcal{OC}^{\succ}_{Sel}$ for reasoning over such relations. Note that this example gives a set of associative compositional axioms which are *not induced* by any precedence $\gg$ on special predicate symbols (see Remark 3.5.5). Hence the chaining calculus formulated in [Bachmair & Ganzinger, 1998*b*] could not be directly applied for such axioms.

Theories of metric distances have recently drawn considerable attention in [Kutz, Wolter, Sturm, Suzuki & Zakharyaschev, 2003; Wolter & Zakharyaschev, 2003], where several extensions of modal logics over distance relations have been proposed. Investigation of such extensions is a useful and interesting direction towards integration of reasoning with geographical data in modal-like formalisms and in particular, in description logics and ontology languages.

Unfortunately the theory of metric distances cannot be completely axiomatised using only compositional axioms. Additional axioms are required for every n:

$$\begin{array}{rlcl} x\,\mathtt{D}_{\leq n}\,x; & \textit{(reflexivity)} & \neg(x\,\mathtt{D}_{\leq n}\,y) \vee \neg(x\,\mathtt{D}_{>n}\,y); & \textit{(disjointness)} \\ x\,\mathtt{D}_{\leq n}\,y \rightarrow y\,\mathtt{D}_{\leq n}\,x; & \textit{(symmetry)} & x\,\mathtt{D}_{\leq n}\,y \vee x\,\mathtt{D}_{>n}\,y; & \textit{(totality)} \end{array} \tag{5.3}$$

(note that the relation $\mathtt{D}_{>n}$ is also symmetric because of *disjointness* and *totality*, and that *reflexivity* together with (5.2) yield monotonicity axioms: $(x\,\mathtt{D}_{<m}\,y \rightarrow x\,\mathtt{D}_{<m+n}\,y)$ and $(x\,\mathtt{D}_{>(m+n)}\,y \rightarrow x\,\mathtt{D}_{>m}\,y)$)

It is possible to embed *reflexivity* and *symmetry* into the chaining calculus by treating the atoms symmetrically and introducing an analog of Reflexivity Resolution rule. For the *disjointness* and *totality* axioms the situation is more difficult, but even in these cases special inference rules could be proposed [see Bachmair & Ganzinger, 1998*b*, Section 6].

The *totality* axiom makes atoms $x\,\mathtt{D}_{\leq n}\,y$ and $x\,\mathtt{D}_{>n}\,y$ not particularly useful as guards. For example, every formula $\forall xy.F[x,y]$ is equivalent to a *guarded* formula $\forall xy.(x\,\mathtt{D}_{\leq n}\,y \rightarrow F[x,y]) \wedge \forall xy.(x\,\mathtt{D}_{>n}\,y \rightarrow F[x,y])$ in presence of the *totality* axiom. Hence the two-variable guarded fragment over such relations becomes as expressible as the two-variable fragment, and hence, is NEXPTIME-hard. This was probably the reason why most known decidable logics based on distance relations [Kutz et al., 2003; Wolter & Zakharyaschev, 2003; Lutz, Wolter & Zakharyaschev, 2003] include only "half" of such relations, namely those of form $x\,\mathtt{D}_{\leq n}\,y$.

Totality can be avoided, if one considers similar relations: $x\,\mathtt{P}_{\leq n}\,y$ and $x\,\mathtt{P}_{>n}\,y$ but with a slightly different semantics. These relations read as: "*there exists a path between x and y of length smaller or equal, respectively greater than n*". We call such relations, the *path relations*. These relations admit the same axioms as the *distance relations* (from (5.2) and (5.3)) except for *disjointness* and *totality*, which makes it easy to integrate them into description logics and the guarded fragment. Moreover, one can extend such relations and express other interesting compositional properties, which do not hold for distance relations, like: $\mathtt{P}_{=m} \circ \mathtt{P}_{=n} \subseteq \mathtt{P}_{=(m+n)}$, or $\mathtt{P}_{[m_1,m_2]} \circ \mathtt{P}_{[n_1,n_2]} \subseteq \mathtt{P}_{[n_1+m_1,n_2+m_2]}$, where $x\,\mathtt{P}_{=n}\,y$, $x\,\mathtt{P}_{[n_1,n_2]}\,y$ mean: "*there exists a path between x and y of the length n, respectively, of the length between n_1 and n_2*".

Relational algebras

In the literature one can find other interesting theories which can be characterised by *extended* of compositional axioms of the following form:

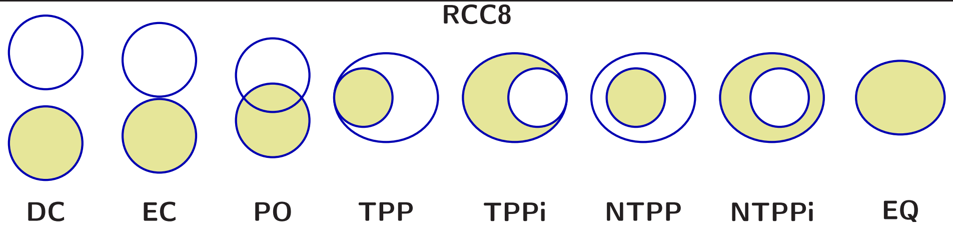
$$S \circ T \subseteq H_1 \cup \cdots \cup H_n \tag{5.4}$$

that is a composition of two relations might be one of *several* other relations. The most prominent examples of such theories are the *region connection calculi* RCC5 and RCC8, and the *Allen's interval algebra*.

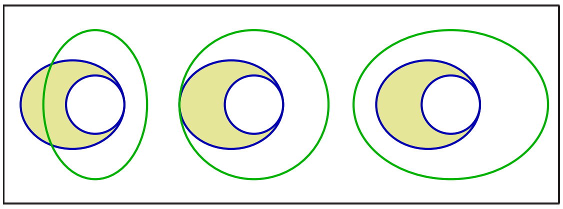
The region connection calculi. The *__r__egion __c__onnection __c__alculi* (short RCC) have been thoroughly studied in the context of *qualitative spatial reasoning* (see [Cohn, Bennett, Gooday & Gotts, 1997] for an overview), and have many potential applications in reasoning about geographical data and in particular within Geographical Information System (GIS).

The RCC-calculi consider *spacial (topological) relations* between *regions*. The RCC8 calculus consists of eight such relations illustrated in Figure 5.1. Here x DC y means that region x is *__d__is__c__onnected* from y, x EC y means that x is *__e__xternally*

Figure 5.1 Spatial relations of RCC8



***c**onnected* with y, x PO y means that x ***p**artially* ***o**verlaps* with y, x TPP y and x NTPP y denotes that x is a ***t**angential*, respectively, ***n**on-**t**angential* ***p**roper* ***p**art* of y (depending on weather the closures of regions intersect or not), x TPPi y and x NTPPi y are their respective inverses, and x EQ y means that two regions are ***eq**al*. These relations are mutually exclusive and cover all possible cases of relations between regions. RCC5 consists of similar five relations between regions when the closures of regions are not taken into account.



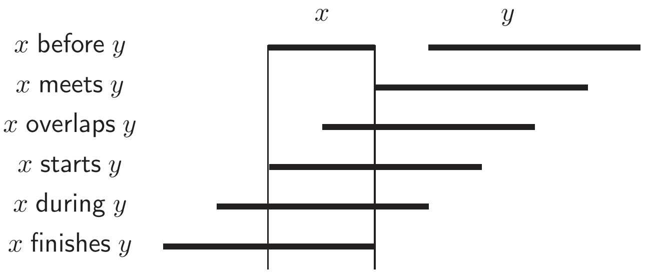
By considering all possible mutual relationships between triples of regions, it is possible to derive all *compositional axioms* for RCC8-relations, which are of form (5.4), for example the following one (see the illustration to the right):

$$\mathsf{TPPi} \circ \mathsf{NTPP} \ \subseteq\ \mathsf{PO} \cup \mathsf{TPP} \cup \mathsf{NTPP}$$

The collection of *all* compositional axioms of such form is called a *compositional table*.

Allen's interval algebra. Another well-known example of a theory that can be characterised by compositional axioms of form (5.4) is the *Allen's interval algebra* [Allen, 1983]. This algebra consists of thirteen relations between *time intervals*: those given in Figure 5.2, their inverses and the equality. Allen's relational algebra can be used in planning for reasoning about processes that have certain duration in time [Allen, 1991]. The compositional table for Allen's interval algebra can be derived similarly by considering all possible relations between end-points of intervals.

Figure 5.2 Allen's [1983] relations between intervals



It is well-known that the Allen's interval algebra and region connection calculi enjoy the properties of *relational algebras*:

Definition 5.1.1. A *relational algebra* is a tuple $\mathcal{A} = (A, \wedge, \neg, \breve{}, \circ, 0, 1, Id)$, where $(A, \wedge, \neg, 0, 1)$ is a *boolean algebra*, $\breve{} : A \to A$ is the *inverse*, $\circ : A \times A \to A$ is a *composition*, $Id \in A$ is the *identity*, such that the following properties hold for every $X, Y, Z \in A$:

$$\begin{array}{lll}
(\mathbf{1})\ X \circ Id = Id \circ X = X & \textit{(identity)} & \\
(\mathbf{2})\ (X \circ Y) \circ Z = X \circ (Y \circ Z) & \textit{(associativity)} & \\
(\mathbf{3})\ (X^{\smile})^{\smile} = X & & \\
(\mathbf{4})\ (X \vee Y^{\smile}) = X^{\smile} \vee Y^{\smile} & & (5.5) \\
(\mathbf{5})\ (\neg X)^{\smile} = \neg(X^{\smile}) & & \\
(\mathbf{6})\ (X \circ Y)^{\smile} = Y^{\smile} \circ X^{\smile} & & \\
(\mathbf{7})\ (X \circ Y) \wedge Z^{\smile} = 0 \ \Rightarrow\ (Y \circ Z) \wedge X^{\smile} = 0 & \textit{(triangle axiom)} & \diamondsuit
\end{array}$$

For an overview on relational algebras see, e.g., [Hirsch, 1997]. Relational algebras that correspond to region and interval calculi are obtained by taking A to be the set of all possible unions of relations, and extending composition and converse on these unions element-wise.

In recent years there have been many attempts for extending modal and description logics with some forms of a spatial reasoning [Haarslev, Lutz & Möller, 1998; Wessel, 2001; Gabelaia, Kontchakov, Kurucz, Wolter & Zakharyaschev, 2003; Lutz & Wolter, 2004] and with interval-based reasoning [Halpern & Shoham, 1991; Lutz, 2004] (see also [Artale & Franconi, 2000] for an overview of such works). However, in many of these papers the spatial and interval relations are interpreted over *restricted* domains. For example Halpern & Shoham [1991] considered only domains consisting of all intervals induced by a *set of points* on the real line; Lutz & Wolter [2004] considered only regions that are *rectangles* in $\mathbb{R}^2$. These restrictions were among the main reasons behind *undecidability* for the resulted logics.

Other papers like [Haarslev et al., 1998; Wessel, 2001] consider classes of theories that overgeneralize spacial theories: for example theories that admit axioms of form (5.4) (and do not fulfil *all* axioms of relational algebras (5.5)). It can be expected that extensions of description logics with such theories are in most cases undecidable (already compositional axioms of form $R \circ S \sqsubseteq H$ are problematic), and only very limited decidability results can be obtained.

In this chapter, among other results, we consider extensions of $\mathcal{ALC}$ with compositional axioms of form (5.4) satisfying *all* properties of relational algebras (5.5). It turns out that even imposing all these "regular" conditions does not help avoiding undecidability results.

5.1.2 A Short History of the Guarded Fragment with Transitivity

On the decision problem and transitivity. Despite its simplicity, the transitivity axiom:

$$\forall xyz.(\, xTy \;\wedge\; yTz \;\rightarrow\; xTz\,) \qquad \textit{(Transitivity)}$$

has always been a challenge for decision procedures in first-order logic, and automated deduction in general. This axiom is expressible in neither *monadic*, nor *two-variable*, or *guarded* fragments which we have studied, because it contains a *binary atom*, *three variables* and *has no guard*. It seems to be that the only well-known decidable fragment that can capture transitivity, is the *Bernays-Schönfinkel class*, which is a set of first-order formulas with a quantifier prefix $\exists^*\forall^*$ [see Börger et al., 1997].

Undecidable extensions of $\mathcal{GF}$ with transitivity. Transitivity is widely used in many logical formalisms, including modal, temporal and description logics – it is indispensable for reasoning about *time* and *ordered structures*. Since the guarded fragment is a most natural first-order counterpart for these formalisms, the reasonable question is: *how to integrate transitivity into into the guarded fragment*? The first *negative* result on this question has been obtained by Grädel [1999]. He showed that already the *three-variable* guarded fragment with *two* transitive relations $\mathcal{GF}^3[\mathsf{transitive}(\mathsf{T}_1, \mathsf{T}_2)]$ is *undecidable*. Later [Ganzinger et al., 1999] improved this results by showing that even *two-variable* guarded fragment with transitivity $\mathcal{GF}^2[\mathsf{T}]$ is undecidable. The reduction proof in the last paper makes use of *five* transitive relations. In the upcoming section we demonstrate that even *two* transitive relations suffice for undecidability of $\mathcal{GF}^2[\mathsf{T}]$.

Decidable extensions of $\mathcal{GF}$ with transitivity. On the positive side, Ganzinger et al. [1999] found a *decidable* variant of the guarded fragment with transitivity which subsumes relational translation of some modal logics over transitive frames, like **K4**—a so called *monadic* two-variable guarded fragment with transitivity $\mathcal{GF}^2_m[\mathsf{T}]$. A *monadic guarded fragment* $\mathcal{GF}_m$ is defined as a set of all guarded formulas, such that every non-unary atom may occur in guards only. In other terms, only *unary atoms* are allowed in the base case of definition (3.17) for guarded formulas (see p. 75). Decidability of $\mathcal{GF}^2_m[\mathsf{T}]$ has been proven in [Ganzinger et al., 1999] by a reduction to the *monadic second-order logic*, which gives only *non-elementary* complexity. Two questions were left open in this paper: (**1**) *Does the guarded fragment remain decidable when only transitive relations are restricted to occur in guards*? and (**2**) *What is the complexity of* $\mathcal{GF}^2_m[\mathsf{T}]$?

The guarded fragment with transitive guards. Two years later, the question (1) above has been answered *positively*: using a model-theoretic analysis, Szwast & Tendera [2001] have demonstrated that the *guarded fragment with transitive guards* (which we denote by $\mathcal{GF}[\mathsf{TG}]$) can be decided in 2EXPTIME. This complexity is optimal, since already the guarded fragment without transitivity $\mathcal{GF}$ is 2EXPTIME-complete. However, the optimal complexity for its *bounded-variable* version $\mathcal{GF}^k[\mathsf{TG}]$, remained unknown, since the procedure of Szwast & Tendera [2001] does not become easy for this case. This complexity gap has been closed by Kieronski [2003], who proved that, surprisingly, $\mathcal{GF}^2[\mathsf{TG}]$ is 2EXPTIME-hard. His proof holds even for the case with *one* transitive relation which is *the only* non-unary atom used in formulas. Hence this result also closes the open question (2) above.

Conclusions. From this short history on the guarded fragment with transitivity, we see, that transitivity is not easy to integrate into decidable fragments: even two-variable guarded fragment becomes undecidable with transitivity, and for its very restricted version, already one transitive relation yields exponential blowup in complexity. Yet, we think that it is impotent to study extensions of decidable fragments with transitivity (both in theory and in practice), because of its fundamental role in computer science.

In the next sections, we sharpen and generalise many results on the guarded fragment with transitivity mentioned here. In particular, we obtain the first *practical* decision procedure for $\mathcal{GF}[\mathsf{TG}]$, using the *ordered chaining calculus*, which has an optimal complexity not only for its full version, but also for its simpler subfragments corresponding to modal and description logics.

5.1.3 Undecidability of the Guarded Fragment with Transitivity

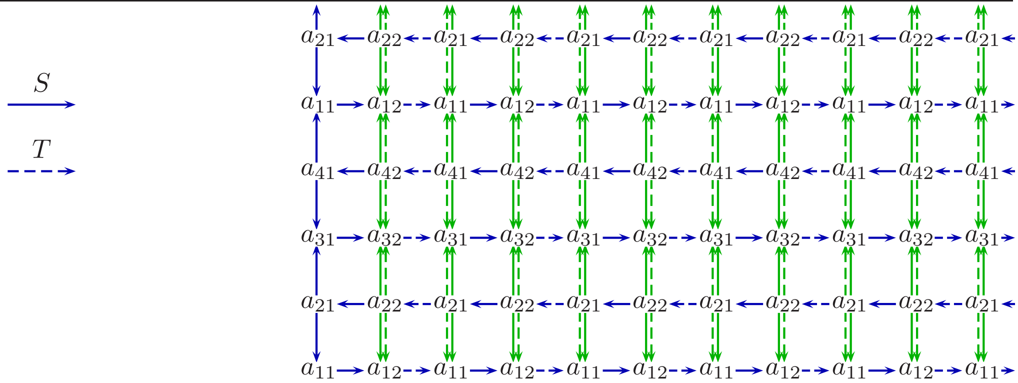
In this section, we demonstrate undecidability for the unrestricted guarded fragment with transitivity $\mathcal{GF}[\mathsf{T}]$. This result has been obtained by Grädel [1999] for $\mathcal{GF}^3[\mathsf{T}]$ with *two* transitive relations and later by Ganzinger et al. [1999] for $\boldsymbol{\mathcal{GF}^2}[\mathsf{T}]$ with *five* transitive relations. Here we sharpen both of these results by proving that $\mathcal{GF}^2[\mathsf{T}]$ with *two* transitive relations is undecidable.

Both undecidability proofs mentioned above use reductions from the tiling problems[2] (see section 3.4), the essential part of which is to enforce an infinite grid structure using guarded formulas with transitivity, similar as it has been done for

[2] Strictly speaking, Ganzinger et al. [1999] use reduction from *Minsky machines*, although there is no essential difference with domino problems

the guarded fragment with functionality $\mathcal{GF}[\mathsf{F}]$ in subsection 4.3.3[3]. We also use reduction from dominoes, but we entail the grid structure in a slightly different way, so that only two variables and only two transitive relations are used.

Figure 5.3 Undecidability of $\mathcal{GF}^2$ with two transitive relations


S
T
a21 a22 a21 a22 a21 a22 a21 a22 a21 a22 a21
a11 a12 a11 a12 a11 a12 a11 a12 a11 a12 a11
a41 a42 a41 a42 a41 a42 a41 a42 a41 a42 a41
a31 a32 a31 a32 a31 a32 a31 a32 a31 a32 a31
a21 a22 a21 a22 a21 a22 a21 a22 a21 a22 a21
a11 a12 a11 a12 a11 a12 a11 a12 a11 a12 a11

The grid that we want to express is illustrated in Figure 5.3. In order to express this structure, we first construct a "skeleton" of a grid: the initial vertical axis, and horizontal lines from it. After that we *propagate* the vertical edges from left to right using transitivity of the respective relations, and special *conditional inclusion axioms* for them:

Table 5.1: Encoding of GRID in the guarded fragment with transitivity

$\mathsf{GRID} \quad := \quad \exists x.[a_{11}(x) \wedge \mathtt{I}(x)] \wedge$ *- creates an origin of a grid*

$\wedge\, \forall x.(\mathtt{I}(x) \wedge a_{11}(x) \rightarrow \exists y.[ySx \wedge \mathtt{I}(y) \wedge a_{21}(y)]\,) \;\wedge$
$\wedge\, \forall x.(\mathtt{I}(x) \wedge a_{21}(x) \rightarrow \exists y.[xSy \wedge \mathtt{I}(y) \wedge a_{31}(y)]\,) \;\wedge$
$\wedge\, \forall x.(\mathtt{I}(x) \wedge a_{31}(x) \rightarrow \exists y.[ySx \wedge \mathtt{I}(y) \wedge a_{41}(y)]\,) \;\wedge$
$\wedge\, \forall x.(\mathtt{I}(x) \wedge a_{41}(x) \rightarrow \exists y.[xSy \wedge \mathtt{I}(y) \wedge a_{11}(y)]\,) \;\wedge$

- creates the initial vertical axis

$\wedge\, \forall x.(a_{11}(x) \rightarrow \exists y.[xSy \wedge a_{12}(y)]) \;\wedge\; \forall x.(a_{12}(x) \rightarrow \exists y.[xTy \wedge a_{11}(y)]) \;\wedge$
$\wedge\, \forall x.(a_{21}(x) \rightarrow \exists y.[ySx \wedge a_{22}(y)]) \;\wedge\; \forall x.(a_{22}(x) \rightarrow \exists y.[yTx \wedge a_{21}(y)]) \;\wedge$
$\wedge\, \forall x.(a_{31}(x) \rightarrow \exists y.[xSy \wedge a_{32}(y)]) \;\wedge\; \forall x.(a_{32}(x) \rightarrow \exists y.[xTy \wedge a_{31}(y)]) \;\wedge$
$\wedge\, \forall x.(a_{41}(x) \rightarrow \exists y.[ySx \wedge a_{42}(y)]) \;\wedge\; \forall x.(a_{42}(x) \rightarrow \exists y.[yTx \wedge a_{41}(y)]) \;\wedge$

- creates all horizontal lines

$\wedge\, \forall xy.[ySx \wedge a_{12}(x) \wedge a_{22}(y) \rightarrow \utilde{yTx}] \;\wedge\; \forall xy.[yTx \wedge a_{11}(x) \wedge a_{21}(y) \rightarrow \utilde{ySx}] \;\wedge$

$\wedge\, \forall xy.[xSy \wedge a_{22}(x) \wedge a_{32}(y) \rightarrow \utilde{xTy}] \;\wedge\; \forall xy.[xTy \wedge a_{21}(x) \wedge a_{31}(y) \rightarrow \utilde{xSy}] \;\wedge$

- propagates the vertical edges

$\wedge\, \forall xy.[xSy \wedge a_{11}(x) \wedge a_{12}(y) \rightarrow H(x,y)] \;\wedge\; \forall xy.[ySx \wedge a_{21}(x) \wedge a_{22}(y) \rightarrow H(x,y)] \;\wedge$

Continued on next page

[3] In fact Grädel [1999] enforces functionality of a binary relation using two transitive relations and reuses this reduction

$$\begin{array}{ll}
\wedge\, \forall xy.[xSy \wedge a_{31}(x) \wedge a_{32}(y) \rightarrow H(x,y)] \;\wedge & \forall xy.[ySx \wedge a_{41}(x) \wedge a_{42}(y) \rightarrow H(x,y)] \;\wedge \\
\wedge\, \forall xy.[xTy \wedge a_{12}(x) \wedge a_{11}(y) \rightarrow H(x,y)] \;\wedge & \forall xy.[yTx \wedge a_{22}(x) \wedge a_{21}(y) \rightarrow H(x,y)] \wedge \\
\wedge\, \forall xy.[xTy \wedge a_{32}(x) \wedge a_{31}(y) \rightarrow H(x,y)] \;\wedge & \forall xy.[yTx \wedge a_{42}(x) \wedge a_{41}(y) \rightarrow H(x,y)] \wedge \\
\wedge\, \forall xy.[ySx \wedge a_{11}(x) \wedge a_{21}(y) \rightarrow V(x,y)] \;\wedge & \forall xy.[xSy \wedge a_{21}(x) \wedge a_{31}(y) \rightarrow V(x,y)] \wedge \\
\wedge\, \forall xy.[ySx \wedge a_{31}(x) \wedge a_{41}(y) \rightarrow V(x,y)] \;\wedge & \forall xy.[xSy \wedge a_{41}(x) \wedge a_{11}(y) \rightarrow V(x,y)] \wedge \\
\wedge\, \forall xy.[yTx \wedge a_{12}(x) \wedge a_{22}(y) \rightarrow V(x,y)] \;\wedge & \forall xy.[xTy \wedge a_{22}(x) \wedge a_{32}(y) \rightarrow V(x,y)] \wedge \\
\wedge\, \forall xy.[yTx \wedge a_{32}(x) \wedge a_{42}(y) \rightarrow V(x,y)] \;\wedge & \forall xy.[xTy \wedge a_{42}(x) \wedge a_{12}(y) \rightarrow V(x,y)] \wedge
\end{array}$$

- *identifies vertical and horizontal edges*

It is crucial in this construction that *distant* nodes do not become "connected" via a relation. In order to prevent this, we interleave two sorts of edges and change their directions. This is important, as the construction does not work, say, for *one* transitive relation, or for two (partial) *equivalence* relations, that is, when transitive relations are additionally symmetric[4]

The conjunction of formula GRID and formula TILING from Figure 3.1 in subsection 4.3.3, is (finitely) satisfiable *iff* the grid admits (periodic) tiling. This implies the following result:

Theorem 5.1.2. $\mathcal{GF}^2[\mathsf{transitive(S,T)}]$ *is a conservative reduction class.*

In fact, it is easy to modify our construction in such a way that S and T are *the only* non-unary atoms used in formulas: for this, atoms $H(x,y)$ and $V(x,y)$ in GRID from Table 5.1, should be replaced with disjunctions that they guard in formula TILING from Figure 3.1.

One possible reason for udecidability of the guarded fragment with transitivity, is a dangerous usage of transitive atoms, when they can be implied by other atoms (we indicated such occurrences in Table 5.1), which made it possible to propagate relations over a grid without connecting too many nodes. It is possible to forbid such occurrences by requiring that transitive atoms should occur in *guard positions only*, as it is the case for translations of many modal and description logics (see the next section). This fragment is called *the guarded fragment with transitive guards* $\mathcal{GF}[\mathsf{TG}]$, and is known to be decidable [Szwast & Tendera, 2001].

Implications for description logics

DL $\mathcal{SHI}$, which is $\mathcal{ALC}$ augmented with *transitive roles*, *role hierarchies* and *inverse roles*, is a *decidable* description logic, contained in a well-studied DL $\mathcal{SHIQ}$

[4] In fact, recently Kieronski & Otto [2005] have demonstrated that the two-variable fragment with two equivalence relations remains decidable

[Horrocks et al., 2000]. It turns out that DL $\mathcal{SI}(\subseteq)$ which is a seemingly harmless generalisation of $\mathcal{SHI}$, with *role hierarchies* replaced by *role-value maps* (see Table 3.2 on p.67) becomes *undecidable.*

Undecidability of description logics incorporating *role-value maps* and *compositions of roles* is known since [Schmidt-Schauß, 1989], [see also Donini, 2003, Section 3.6.1]. However, it is generally understood that this undecidability is caused by *compositions of roles*, since already many *simple* description logics *without role-value maps* become undecidable with composition: see [Baader, 2003] and section 2.5. All these undecidability proofs do not work for *transitive* roles. It comes to our surprise that *transitivity* with *role-value maps* may also form a dangerous mixture:

Corollary 5.1.3. *Subsumption and satisfiability of concepts in DL $\mathcal{SI}(\subseteq)$ is undecidable.*

Proof. First, we prove that a concept satisfiability w.r.t. $\mathcal{SI}(\subseteq)$-TBoxes is undecidable. For this, we express formula GRID from Table 5.1 using a concept and inclusion axioms.

The initial "skeleton" can be enforced by saying that a concept $a_{11} \sqcap \mathtt{I}$ that corresponds to the first conjunct $\exists x.[a_{11}(x) \wedge \mathtt{I}(x)]$ in GRID is satisfiable w.r.t. a TBox containing the following inclusion axiom for subsequent conjuncts from GRID:

$$\forall x.(\mathtt{I}(x) \wedge a_{11}(x) \to \exists y.[ySx \wedge \mathtt{I}(y) \wedge a_{21}(y)]\,) \dashrightarrow \mathtt{I} \sqcap a_{11} \sqsubseteq \exists S^{-}.(\mathtt{I} \sqcap a_{21})$$

$$\forall x.(\mathtt{I}(x) \wedge a_{21}(x) \to \exists y.[ySx \wedge \mathtt{I}(y) \wedge a_{31}(y)]\,) \dashrightarrow \mathtt{I} \sqcap a_{21} \sqsubseteq \exists S^{-}.(\mathtt{I} \sqcap a_{31})$$

$$\ldots\ldots\ldots \textit{etc.}$$

$$\forall x.(a_{11}(x) \to \exists y.[xSy \wedge a_{12}(y)]) \dashrightarrow a_{11} \sqsubseteq \exists S.a_{12}$$

$$\forall x.(a_{12}(x) \to \exists y.[xTy \wedge a_{11}(y)]) \dashrightarrow a_{12} \sqsubseteq \exists T.a_{11}$$

$$\ldots\ldots\ldots \textit{etc.}$$

where $\mathtt{I}$ and a_{ij}, $1 \leq i \leq 4$, $j = 1, 2$ are *concept names* that correspond to the unary atoms, and S and T are *role names* which are declared to be transitive.

It is more difficult to express the remaining conjuncts in GRID which are *conditional* inclusion axioms used for propagation of vertical edges and defining relations H and V. Although these axioms are not expressible using *role hierarchies*, it is possible to express them using *role-value maps* as follows:

$$\forall xy.[ySx \wedge a_{12}(x) \wedge a_{22}(y) \to yTx] \dashrightarrow a_{12} \sqsubseteq (S^{-} \subseteq T_{22})$$

$$\dashrightarrow a_{22} \sqsubseteq (T_{22}^{-} \subseteq T^{-})$$

$$\ldots\ldots\ldots \textit{etc.}$$

$$\forall xy.[xSy \wedge a_{11}(x) \wedge a_{12}(y) \to H(x,y)] \dashrightarrow a_{11} \sqsubseteq (S \subseteq H_{12})$$

$$\dashrightarrow a_{12} \sqsubseteq (H_{12}^{-} \subseteq H^{-})$$

$$\ldots\ldots\ldots \textit{etc.}$$

Here T_{22},.., H_{12},.. are auxiliary role names (*not* declared to be transitive). Formula TILING for a domino problem can be easily encoded by inclusion axioms (even in $\mathcal{ALC}$). Hence a formula $\mathsf{GRID} \wedge \mathsf{TILING}$ is satisfiable *iff* the concept $a_{11} \sqcap \mathtt{I}$ is satisfiable w.r.t. the constructed TBox. This proves that a concept satisfiability *w.r.t. a terminology* is undecidable for $\mathcal{SI}(\subseteq)$.

In order to demonstrate undecidability of just concept satisfiability (without TBox-es), we note that any $\mathcal{SI}(\subseteq)$-TBox can be *internalised* (that is, imbedded into a concept). This can be done using an additional *transitive role* U and role-value maps:

- *for every TBox inclusion axiom* $C_1 \sqsubseteq C_2$ *we add a conjunct* $\forall U.(\neg C_1 \sqcup C_2)$*;*
- *for every role name* R *we add a conjunct*
$$(R \subseteq U) \sqcap (R^- \subseteq U) \sqcap \forall U.[(R \subseteq U) \sqcap (R^- \subseteq U)].$$

The conjuncts of the last form express that U is a *universal role*, i.e., it contains any other role and its converse. If U is additionally declared to be transitive, then $\forall U.C$ expresses that C holds *globally*, i.e., it is equivalent to inclusion axiom $\top \sqsubseteq C$. The *internalisation* technique is well-known for expressive description logics [see Baader, Calvanese, McGuinness, Nardi & Patel-Schneider, 2003].

Finally, concept subsumption in $\mathcal{SI}(\subseteq)$ is undecidable, since $\bot$ subsumes C *iff* C is not satisfiable. ⧉

5.1.4 On the Modal Fragment with Transitivity

As we have seen in the previous section, the guarded fragment with transitivity remains undecidable even when restricted to *two variables* and to *two transitive relations.* However many modal and description logics which could be translated to *two-variable guarded fragment*, remain decidable with transitivity. In fact, transitivity does not even change their *complexity.* So, which distinguished properties of these logics are responsible for this?

In this section we try to approach this question, by considering the extension **K4** of the basic modal logic **K** with *transitivity.* Modal logic **K4** consists of modal formulas that are valid in all *transitive frames* (i.e., the set of admissible Kripke interpretations for **K4** is restricted to those where the reachability relation R is transitive: see subsection 3.2.1). As any modal logic with a first-order expressible *frame correspondence property* (like those listed in Table 3.1 on p.64), validity (and dually, satisfiability) of **K4**-formulas can be decided through the first-order logic via a *relational translation* for modal formulas (3.5). We reformulate this translation, using our standard sequence of transformations:

Starting from a recursive definition for modal formulas:

$$\mathrm{MF} \;::=\; A \mid \neg F_1 \mid F_1 \wedge F_2 \mid \Box F_1\ . \tag{5.6}$$

we put them into the *negation-normal form*:

$$[\mathrm{MF}]^{nnf} \;::=\; (\neg)A \mid F_1 \mathbin{\wedge\!\!\!\!\vee} F_2 \mid \Box F_1 \mid \Diamond F_1. \tag{5.7}$$

and apply the *structural transformation*: $[F]^{str}_{\mathbf{K}} := \mathtt{P}_F \wedge [F]^{def}_{\mathbf{K}}$, where:

$$\begin{aligned}
[F]^{def}_{\mathbf{K}} := [(\neg)A]^{def}_{\mathbf{K}} &= \forall x.[\mathtt{p}_F(x) \rightarrow (\neg)A(x)] & \mid\\
[F_1 \mathbin{\wedge\!\!\!\!\vee} F_2]^{def}_{\mathbf{K}} &= \forall x.[\mathtt{p}_F(x) \rightarrow \mathtt{p}_{F_1}(x) \mathbin{\wedge\!\!\!\!\vee} \mathtt{p}_{F_2}(x)] \wedge [F_1]^{def}_{\mathbf{K}} \wedge [F_2]^{def}_{\mathbf{K}} & \mid\\
[\Box F_1]^{def}_{\mathbf{K}} &= \forall x.(\mathtt{p}_F(x) \rightarrow \forall y.[(xRy) \rightarrow \mathtt{p}_{F_1}(x)]) \wedge [F_1]^{def}_{\mathbf{K}} & \mid\\
[\Diamond F_1]^{def}_{\mathbf{K}} &= \forall x.(\mathtt{p}_F(x) \rightarrow \exists y.[(xRy) \wedge \mathtt{p}_{F_1}(x)]) \wedge [F_1]^{def}_{\mathbf{K}}\ .
\end{aligned} \tag{5.8}$$

Then a **K4**-formula F is satisfiable *iff* the first-order formula

$$[F]^{str}_{\mathbf{K}} \;\wedge\; \forall xyz.[xRy \wedge yRz \rightarrow xRz]$$

is satisfiable. It is easy to see from (5.8), that $[F]^{str}_{\mathbf{K}}$ is a two-variable *guarded formula*, however the second conjunct expressing the *transitivity axiom* is not. Since the target fragment of this translation – the two-variable guarded fragment with transitivity – is undecidable, this translation can be hardly used for deciding **K4**.

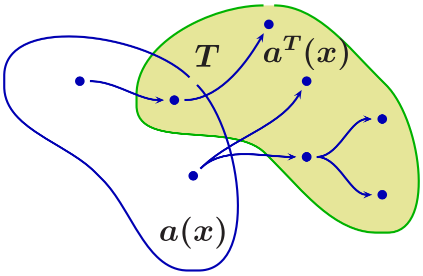
de Nivelle [1999] has proposed an alternative translation, that maps **K4** (and many other modal logics) *directly* into the guarded fragment. This translation is obtained from (5.8), by adding one more conjunct to the case with $\Box F_1$:

$$\begin{aligned}
[F]^{def}_{\mathbf{K4}} := [(\neg)A]^{def}_{\mathbf{K4}} &= \forall x.[\mathtt{p}_F(x) \rightarrow (\neg)A(x)] & \mid\\
[F_1 \mathbin{\wedge\!\!\!\!\vee} F_2]^{def}_{\mathbf{K4}} &= \forall x.[\mathtt{p}_F(x) \rightarrow \mathtt{p}_{F_1}(x) \mathbin{\wedge\!\!\!\!\vee} \mathtt{p}_{F_2}(x)] \wedge [F_1]^{def}_{\mathbf{K4}} \wedge [F_2]^{def}_{\mathbf{K4}} & \mid\\
[\Box F_1]^{def}_{\mathbf{K4}} &= \boxed{\forall x.(\mathtt{p}_F(x) \rightarrow \forall y.[(xRy) \rightarrow \mathtt{p}_F(x)])} \wedge &\\
&\quad\ \forall x.(\mathtt{p}_F(x) \rightarrow \forall y.[(xRy) \rightarrow \mathtt{p}_{F_1}(x)]) \wedge [F_1]^{def}_{\mathbf{K4}} & \mid\\
[\Diamond F_1]^{def}_{\mathbf{K4}} &= \forall x.(\mathtt{p}_F(x) \rightarrow \exists y.[(xRy) \wedge \mathtt{p}_{F_1}(x)]) \wedge [F_1]^{def}_{\mathbf{K4}}\ .
\end{aligned} \tag{5.9}$$

It is proved that a *guarded formula* $[F]^{str}_{\mathbf{K4}} := \mathtt{P}_F \wedge [F]^{def}_{\mathbf{K4}}$ is satisfiable *iff* F is satisfiable in **K4**.

Under a closer inspection one could notice that translations (5.8) and (5.9), naturally correspond to the tableau *expansion rules* given in System 1. In particular, the case for $\Box F_1$ from (5.9) encode respectively the *expansion rules* `4` and `K`.

We have analysed the de Nivelle's [1999] translation (5.9) from a viewpoint of the *clause logic*, and came up with the following rule:



Transitive Closure

$$\texttt{TC}: \frac{\neg(xTy) \vee \neg a(x) \vee b(y)}{\begin{array}{r}\neg(xTy) \vee \neg a(x) \vee a^T(y) \\ \neg(xTy) \vee \neg a^T(x) \vee a^T(y) \\ \neg a^T(y) \vee b(y)\end{array}}$$

$$\left[\begin{array}{l}\textit{where (i) } xTy \textit{ is a transitive relation,} \\ \textit{and (ii) } a^T(x) \textit{ is a fresh unary atom} \\ \textit{for introduced for } a \textit{ and } T.\end{array}\right] \tag{5.10}$$

This rule extends a signature with an atom $a^T(x)$, which intuitively represents the "image" of $a(x)$ under relation T – this is what the first clause in the conclusion of (5.10) expresses. If T is a transitive relation, then a^T is closed under T, which is expressed by the second clause. Finally, every element of $a^T(y)$ is an element of $b(y)$, because the premise of this rule says that every element y that is T-reachable from some x where $a(x)$ holds (i.e., exactly those from $a^T(y)$) should be in $b(y)$. In fact, we have just proved the following lemma:

Lemma 5.1.4. *Transitive Closure is a sound inference rule.*

Proof. Let $\mathcal{M}$ be a T-model for the premise $\neg(xTy) \vee \neg a(x) \vee b(y)$ of rule (5.10) (i.e., a model of this formula in which T is interpreted by a transitive relation). We expand $\mathcal{M}$ to a model $\mathcal{M}'$ by interpreting the new atom $a^T(x)$ as follows:

$$[a^T(y)]^{\mathcal{M}'} := [\exists x.(a(x) \wedge xTy)]^{\mathcal{M}}.$$

This definition (when read from right to the left) implies, in particular that:

$$\mathcal{M}' \models [\exists x.(a(x) \wedge xTy)] \rightarrow a^T(x)$$

which means that the first conclusion of rule (5.10) is `true` in $\mathcal{M}'$. The second conclusion is also `true` in $\mathcal{M}'$ because of the following sequence of implications:

$$\begin{array}{c}\mathcal{M}' \models \underline{a^T(x) \wedge xTy} \;\equiv\; \exists x'.[a(x') \wedge x'Tx \wedge xTy] \;\rightarrow \\ (\text{transitivity of } T \Rightarrow) \;\; \exists x'.[a(x') \wedge x'Ty] \;\equiv\; \underline{a^T(y)}.\end{array}$$

The last conclusion of (5.10), is implied by the premise of this rule:

$$\mathcal{M}' \models \underline{a^T(y)} \;\equiv\; \exists x.[a(x) \wedge xTy] \overset{\text{premise}}{\longrightarrow} \exists x.b(y) \;\equiv\; \underline{b(y)}.$$

□

Since the Transitive Closure rule is sound, it can be used in a saturation process as an optional *simplification rule*. However, this rule does not really produce "simpler clauses". A really useful feature of this rule is that an atom xTy may no longer assumed to be transitive, when all clauses of form:

$$\neg(xTy) \vee \neg a(x) \vee b(y) \tag{5.11}$$

are "closed" under the Transitive Closure rule:

Lemma 5.1.5. *Let N be a set of clauses containing transitive atoms negatively only in clauses of the form* (5.11)*. Let N' be obtained from N by adding the conclusions of the Transitive Closure rule for all of these clauses. Then N is T-satisfiable iff N' is satisfiable.*

Proof. The "*only if*" part of this lemma follows from soundness of the Transitive Closure rule. Indeed, if N is T-satisfiable, then by Lemma 5.1.4, N' is also T-satisfiable, and hence N' is satisfiable.

For proving the "*if*" part, suppose $\mathcal{M}'$ is a model of N'. We need to construct a T-model $\mathcal{M}$ for N. We simply define $\mathcal{M}$ to be the *smallest* T-interpretation that contains $\mathcal{M}'$, i.e., $\mathcal{M}$ is identical to $\mathcal{M}'$ for all non-transitive atoms, whereas interpretations for transitive atoms are *transitively closed*: $T^{\mathcal{M}} := (T^{\mathcal{M}'})^+$. We claim that the clauses from N remain `true` in $\mathcal{M}$:

Obviously, the clauses containing transitive atoms *positively* remain `true` in $\mathcal{M}$. The remaining clauses from N must be of the form (5.11). Such a clause may become `false` in $\mathcal{M}$, only if there are some domain elements d_i with $0 \leq i \leq n$, such that: $(\boldsymbol{i})$ $\mathcal{M}' \vDash a(d_0)$, $(\boldsymbol{ii})$ $\mathcal{M}' \vDash d_{i-1} T d_i$, $1 \leq i \leq n$, but $(\boldsymbol{iii})$ $\mathcal{M}' \nvDash b(d_n)$. However, since the conclusions of these clauses under the Transitive Closure rule, are `true` in $\mathcal{M}'$, we have in particular:

$$\begin{array}{ll}
(\mathbf{1})\ \mathcal{M}' \vDash & \neg(d_0 T d_1) \vee \neg a(d_0) \vee a^T(d_1); \\
(\mathbf{2})\ \mathcal{M}' \vDash & \neg(d_{i-1} T d_i) \vee \neg a^T(d_{i-1}) \vee a^T(d_i),\ 1 \leq i \leq n; \\
(\mathbf{3})\ \mathcal{M}' \vDash & \neg a^T(d_n) \vee b(d_n);
\end{array}$$

which is not possible together with (i) – (iii). Hence every clause from N is true in $\mathcal{M}$. ⧉

To summarise, we have shown how transitivity can be effectively eliminated for certain clause classes, containing translations of modal formulas, using an additional inference rule (5.10). However, this solution is not really satisfactory, since it relies on *global properties* of a clause set: *all negative occurrences of transitive atoms must be of form* (5.11). This makes application of this rule rather limited: if a clause

$\neg(xTy) \vee \neg a(x) \vee b(y)$ is replaced with an essentially equivalent pair of clauses $\neg(xTy) \vee \neg a(x) \vee b(y) \vee c(x,y)$ and $\neg c(x,y)$, our arguments stop working.

In the next section we demonstrate how to fix this problem and integrate the Transitive Closure rule into our general theorem proving framework.

5.2 Extensions without Equality

In this section we consider extensions of the guarded fragment without equality by compositional axioms. First we demonstrate how to obtain a saturation-based decision procedure for the *guarded fragment with transitive guards* $\mathcal{GF}[\mathsf{TG}]$ without equality. Then we extended this procedure for arbitrary *associative compositional* axioms instead of transitivity, and discuss how restrictions on occurrences of special (transitive or compositional) atoms can be relaxed. Finally, we demonstrate where is the limit of such extensions: the two-variable guarded fragment over a *relational algebra* is undecidable even restricted to the *modal fragment*.

5.2.1 Deciding the Guarded Fragment with Transitive Guards

For deciding the *guarded fragment with transitive guards* $\mathcal{GF}[\mathsf{TG}]$ without equality, we employ the *ordered chaining calculus* $\mathcal{OC}^{\succ}_{Sel}$ described in subsection 3.5.4, which has build-in inferences for transitive relations. In [Kazakov & de Nivelle, 2004] we have demonstrated how $\mathcal{GF}[\mathsf{TG}]$ without equality can be decided by resolution with *ordering constraints*. However, using the chaining calculus, constraint clauses can be avoided, which means that our procedure could be, in principle, extended to the case with equality.

The chaining calculus was proven to perform well in practice on problems involving transitivity [Bachmair & Ganzinger, 1998*b*]. It is also able do decide some modal logics with transitivity [Ganzinger et al., 2001] (although, see Note 5.2.2 below). Unfortunately, the chaining calculus does not decide directly the clause class for $\mathcal{GF}[\mathsf{TG}]$. A problematic situation may appear which will be described in a moment. After spotting this problem, we propose a solution, using an additional simplification rule Transitive Closure considered in the previous sections.

Difficulties with negative occurrences of transitive atoms

When we translate formulas from $\mathcal{GF}[\mathsf{TG}]$ to clauses according to our general procedure, we obtain clauses containing transitive atoms only of the following forms:

$$\begin{array}{ll} (\mathbf{1})\ \neg p(x) \vee h(x)Tx; & (\mathbf{4})\ \neg(xTy) \vee \alpha[x,y]; \\ (\mathbf{2})\ \neg p(x) \vee xTh(x); & (\mathbf{5})\ \neg(xTx) \vee \alpha[x]; \\ (\mathbf{3})\ \neg p(x) \vee xTx; & \end{array} \tag{5.12}$$

(the exact clause types are similar to those for $\mathcal{GF}[\mathsf{FG}]$: see Table 4.18 on p.140).

Chaining rules (see System 6 on p.84) do solve many problems with transitivity axiom (in particular those mentioned in section 2.3 and in [Ganzinger et al., 2001]), but not all of them. It can be noticed that clauses of forms (1) – (3), with positive occurrences of transitive atoms behave "well" under the chaining inferences: the Ordered Chaining rule applied to them does not increase the *number of variables* in clauses (since there is only one variable), and does not increase their *functional depth*, since the deepest terms are unified. There are also no difficulties with the Negative Chaining rule applied to clauses (1) – (3) and a clause of form (5). However, Negative Chaining with clauses of form (4) may cause problems:

Example 5.2.1. Consider the following clauses of forms (1), (2) and (4):

$$\begin{array}{ll} 1.\ \neg p(x) \vee \underline{\boldsymbol{h(x)}}\boldsymbol{Tx}^{\star} & 3.\ \boldsymbol{\neg(xT\underline{y})} \vee \neg a(x) \vee a(y) \\ 2.\ \neg p(x) \vee \boldsymbol{xT}\underline{\boldsymbol{h(x)}}^{\star} & 4.\ \boldsymbol{\neg(xT\underline{y})} \vee a(x) \vee \neg a(y) \end{array}$$

Clearly, the Negative Chaining inferences between clauses 1, 2 and 3, 4 are possible:

$\mathtt{NC}[2;3]$: 5. $\neg p(y) \vee \neg(xTy) \vee \neg a(x) \vee a(\underline{h(y)})$

$\mathtt{NC}[2;4]$: 6. $\neg p(y) \vee \neg(xTy) \vee a(x) \vee \neg a(\underline{h(y)})$

One can already see that clauses 5 and 6 do not look good: they contain a functional term $h(y)$ that is *not covering* for a clause, since it does not contain the variable x. This may potentially result in increase of the number of variables in clauses. And this happens indeed: unless no other literals are selected in clauses, a resolution inference is possible that resolves the last literals:

$\mathtt{OR}[5;6]$: 7. $\neg p(y) \vee \neg(x_1Ty) \vee \neg(x_2Ty) \vee \neg a(x_1) \vee a(x_2)$

(we have eliminated duplicate occurrences of $\neg p(y)$). Clause 7 is similar to clauses 2 and 3 but is longer. It is possible to show that arbitrary long clauses can be produced in this way by continuing Negative Chaining inferences with clauses 1 and 2 followed by Ordered Resolution.

On the other hand, if we select some negative literals in clauses 5 and 6 to prevent their resolving, then the *clause depth* can grow, for example if one selects the negative literal $\neg(xTy)$, then the Negative Chaining inference with clause 2 would produce similar but deeper clauses. ⟐

Note 5.2.2. For the clause class correspondent to extensions of modal logic **K4** *considered in [Ganzinger et al., 2001], the problem illustrated in Example 5.2.1 can be partially solved. The authors have demonstrated that "long" clauses can be simplified using a special Condensing rule. This makes it possible to establish a bound on the size of such clauses and regain termination of the saturation procedure. However the clauses that can be obtained after condensation could be exponentially long in the size of the signature, which gives only 2EXPTIME decision procedure for* **K4** *(which is "only" PSPACE-complete). The authors pointed out that this problem could be avoided by splitting long clauses on shorter ones using Splitting through New Predicate Symbol.* ◈

More examples

Let us analyse the problematic situation illustrated in Example 5.2.1.

Example 5.2.3 [Example 5.2.1, continued]. Negative Chaining is not the only rule that can be applied to clauses 2 and 3, respectively 2 and 4. These clauses can be also resolved:

$\mathtt{OR}[2;3]$: $5'.\ \neg p(x) \vee \neg\underline{a(x)} \vee a(h(x))$
$\mathtt{OR}[2;4]$: $6'.\ \neg p(x) \vee \underline{a(x)} \vee \neg a(h(x))$

Note that clauses 5 and 6 can be now obtained by resolving these clauses with clauses 3 and 4 on $a(y)$ and $\neg a(y)$ respectively. However, since these literals are *not maximal* in these clauses, the resolution inferences should not be made. The fact that the conclusion of some inference can be obtained by an "unordered" resolution with some other clauses, is a strong indicator that this inference might be *redundant*.

Recall (see subsection 3.5.2) that (***i***) a *ground* inference is *redundant* if the conclusion of this inference follows from clauses that are *smaller* in the ordering than the maximal premise of this inference, and (***ii***) a *non-ground* inference is *redundant*, if every ground instance of this inference is redundant. Redundancy of inferences can justify many useful refinements of saturation procedures, notably, that paramodulation into variables are not needed (see [Kazakov, 2005] for further details).

Let us demonstrate formally, that inferences $\mathtt{NC}[2;3]$ and $\mathtt{NC}[2;4]$ are redundant indeed. Any ground instance of the first inference has the form:

$2_0.\ \neg p(t) \vee \boldsymbol{tT\underline{h(t)}}^{\star}$ $\quad 3_0.\ \boldsymbol{\neg(sT\underline{h(t)})} \vee \neg a(s) \vee a(h(t))$
$\mathtt{NC}[2_0;3_0]$: $5_0.\ \neg\underline{p(t)} \vee \neg(sTt) \vee \neg\underline{a(s)} \vee a(h(t))$

As we have pointed out above, clause 5 can be obtained by resolving clauses $5'$ and 3, so its ground instance 5_0 follows from the following ground instances of these clauses:

$5'_0.\ \neg p(t) \vee \neg \underline{a(t)} \vee a(h(t)) \qquad 3_1.\ \neg(sTt) \vee \neg a(s) \vee \underline{a(t)}$
$\quad \Rightarrow \quad : 5_0.\ \neg p(t) \vee \neg(sTt) \vee \neg a(s) \vee a(h(t))$

It remains to notice that both clauses $5'_0$ and 3_1 are *smaller in the ordering* than, say, clause 3_0 used in the first inference above. Hence, inference $\mathtt{NC}[2;3]$ is redundant. Similarly, one can show that inference $\mathtt{NC}[2;4]$ is redundant as well. ◇

Example 5.2.3 demonstrates that "dangerous" Negative Chaining inferences can be sometimes avoided provided that some "safe" Ordered Resolution inferences are made. Unfortunately the construction from this example does not work for *all* clauses of form (4) from (5.12).

Example 5.2.4 [Example 5.2.3, continued].

Let us try to repeat the procedure from Example 5.2.3, for other clauses:

$2.\ \neg p(x) \vee \boldsymbol{xT\underline{h(x)}}^{\star} \qquad 3.\ \boldsymbol{\neg(xT\underline{y})} \vee \neg a(x) \vee \underline{b(y)},\ \textit{where } a \neq b$
$\mathtt{NC}[2;3]\colon 5.\ \neg p(y) \vee \neg(xTy) \vee \neg a(x) \vee b(h(y))$
$\mathtt{OR}[2;3]\colon 5'.\ \neg p(x) \vee \neg \underline{a(x)} \vee b(h(y))$

We see that clauses 3 and $5'$ do not resolve anymore to produce 5. So, how to be? Surprisingly, a solution can be found using the Transitive Closure rule (5.10) considered in subsection 5.1.4. If we apply this rule to clause 3 above, we obtain new clauses:

$3_a.\ \boldsymbol{\neg\underline{(xTy)}} \vee \neg a(x) \vee \underline{a^T(y)}$
$3_b.\ \boldsymbol{\neg\underline{(xTy)}} \vee \neg a^T(x) \vee \underline{a^T(y)}$
$3_c.\ \neg\underline{a^T(y)} \vee b(y)$

By resolving clause 2 with first two clauses we obtain clauses:

$5'_a.\ \neg p(x) \vee \neg a(x) \vee a^T(h(x))$
$5'_b.\ \neg p(x) \vee \neg \underline{a^T(x)} \vee \underline{a^T(h(x))}$

Now clause 5 above follows from clauses 3_a, $5'_b$ and 3_c by resolving on respective literals, which similarly implies that inference $\mathtt{NC}[2;3]$ producing clause 5 is *redundant*. Moreover, the results of Negative Chaining inferences with newly generated clauses:

$\mathtt{NC}[2;3_a]\colon 5_a.\ \neg p(y) \vee \neg(xTy) \vee \neg a(x) \vee a^T(h(y))$
$\mathtt{NC}[2;3_b]\colon 5_b.\ \neg p(y) \vee \neg(xTy) \vee \neg a^T(x) \vee a^T(h(y))$

can be also produced by resolving clauses 3_a with $5'_b$ and 3_b with $5'_b$, respectively, and hence these inferences are also redundant. So further Transitive Closure inferences for clauses $5'_a$ and $5'_b$ are not needed and all Negative Chaining inferences are effectively blocked. ◇

Redundancy of Negative Chaining inferences

Now the rôle of the Transitive Closure rule in a saturation procedure becomes clear. This rule, unlike other *simplification rules*, does not make its premise *redundant*, but *it makes some inferences with the premise redundant.* So, it is a new kind of simplification rules. Now, when everything seems to become clear, let us formalise the ideas described above.

First, we generalise the Transitive Closure rule introduced in subsection 5.1.4 to a wider class of clauses: see Figure 5.4. Here the rule is applied to a clause containing one negative occurrence of a transitive atom over distinct variables x and y, the rest of which can be split into variable disjoint parts over variables x and y. An *extended* predicate symbol u^T_α is introduced now not for a unary literal containing x as before, but for a *subclause* $\alpha[x]$ of all such literals in the premise. Apart from this, there are no essential differences with the rule that we have considered in subsection 5.1.4. In particular, soundness for this rule can be shown analogously to Lemma 5.1.4.

Figure 5.4 The general Transitive Closure rule

Transitive Closure

$$\texttt{TC}: \frac{\boldsymbol{\neg(xTy)}^\sharp \vee \alpha[x] \vee \beta[y]}{\begin{array}{r} \neg(xTy) \vee \alpha[x] \vee u^T_\alpha(y) \\ \neg(xTy) \vee \neg u^T_\alpha(x) \vee u^T_\alpha(y) \\ \neg u^T_\alpha(y) \vee \beta[y] \end{array}}$$

[*where (i) T is a transitive predicate; (ii) u^T_α is an extended unary predicate symbol introduced for α and T*]

In order to prove redundancy of Negative Chaining inferences, we need to make an additional assumption about the ordering $\succ$ of our chaining calculus $\mathcal{OC}^{\succ}_{Sel}$:

Definition 5.2.5. We say that an ordering $\succ$ is *compatible with arities of special predicate symbols*, (short *CASP*) if: $b[!t_1, !t_2] \succ \neg(t_1 S t_2) \succ (t_1 S t_2) \succ \neg u(t_1)$ for any *non-special* predicate symbol b with $ar(b) \geq 2$, *special* predicate symbol S and a unary predicate symbol u. ◈

CASP-orderings can be found by taking any total reduction ordering $\succ$ on ground terms and comparing literals according to the following complexity measure:[5]

$$c(L) \;:=\; (\, max(L)\, ,\, 1{-}pol(L)\, ,\, ar(L)\, ,\, s(L)\, ,\, min(L)\,) \qquad (5.13)$$

where $max(L)$ and $min(L)$ are respectively, the maximal and the minimal arguments of L, $pol(L)$ is the polarity of L (1 – *positive*, 0 – *negative*), $ar(L)$ is the arity of the predicate symbol of L, $s(L) = 0$ if L contains a special predicate symbol and, $s(L) = 1$ otherwise. This ordering is *simple* (see Definition 4.1.4), *CAP* (see

[5] Such ordering could always be extended to a *total* ordering on ground literals

Definition 4.1.9) and *CASP*. It is a routine to check that this ordering is *admissible for chaining* according to Definition 3.5.6 on p.85

From now on we assume that the *ordered chaining calculus* $\mathcal{OC}^{\succ}_{Sel}$ is parametrised with a *simple* ordering $\succ$ which is *CAP* and *CASP*. Now we give our main lemma, which summarises the purpose of the Transitive Closure rule:

Lemma 5.2.6. *Let T be a transitive predicate symbol and N be a clause set such that the following clauses are contained in N or redundant w.r.t. N:*

1. $C \vee \boldsymbol{sTt}$, $\quad$ $\mathtt{R}_0$. $\neg\boldsymbol{(xTy)}^{\sharp} \vee \alpha[x] \vee \beta[y]$,
$\mathtt{R}_1$. $\neg\boldsymbol{(xTy)} \vee \alpha[x] \vee u(y)$ $\quad$ $\mathtt{OR}[1;\mathtt{R}_1]$: 2_a. $C \vee \alpha[s] \vee u(t)$
$\mathtt{R}_2$. $\neg\boldsymbol{(xTy)} \vee \neg u(x) \vee u(y)$ $\quad$ $\mathtt{OR}[1;\mathtt{R}_2]$: 2_b. $C \vee \neg u(s) \vee u(t)$
$\mathtt{R}_3$. $\neg u(y) \vee \beta[y]$;

Then the following Negative Chaining *inferences are redundant w.r.t. N:*

($\boldsymbol{a}$) $\mathtt{NC}[1;\mathtt{R}_0]$ *(left)* : 3_a. $C \vee \neg(tTy) \vee \alpha[s] \vee \beta[y]$
($\boldsymbol{b}$) $\mathtt{NC}[1;\mathtt{R}_0]$ *(right)*: 3_b. $C \vee \neg(xTs) \vee \alpha[x] \vee \beta[t]$

Proof. The proof of this lemma can be found in Appendix D.1. □

Lemma 5.2.7. *Let N be a clause set containing clauses:*
1. $C \vee \boldsymbol{sTt}$, $\quad$ 2. $\neg\boldsymbol{(xTy)}^{\sharp} \vee \alpha[x] \vee \beta[y]$,

and the conclusions of the following inferences:

$\mathtt{TC}[\mathtt{R}_0]$: 3. $\neg\boldsymbol{(xTy)} \vee \alpha[x] \vee u^T_\alpha(y)$ $\quad$ $\mathtt{OR}[1;3]$: $C \vee \alpha[s] \vee u^T_\alpha(t)$
4. $\neg\boldsymbol{(xTy)} \vee \neg u^T_\alpha(x) \vee u^T_\alpha(y)$ $\quad$ $\mathtt{OR}[1;4]$: $C \vee \neg u^T_\alpha(s) \vee u^T_\alpha(t)$
5. $\neg u^T_\alpha(y) \vee \beta[y]$;

Then all Negative Chaining *inferences between clause* 1 *and clauses* 2*,* 3 *and* 4 *are redundant.*

Proof. Redundancy of Negative Chaining inferences with clause 2 follows directly from Lemma 5.2.6 by taking $u(x) := u^T_\alpha(x)$.

In order to prove redundancy for clause 3, we apply this lemma for $\beta(y) := u^T_\alpha(y)$, and $\mathtt{R}_1$, $\mathtt{R}_2$ and $\mathtt{R}_3$ be respectively clauses 3, 4, and $\neg u^T_\alpha(y) \vee u^T_\alpha(y)$. Since the first two clauses are in N and the last clause is redundant w.r.t. N, by Lemma 5.2.6 we have that the Negative Chaining inference $\mathtt{NC}[1;3]$ is redundant.

For proving redundancy of inference $\mathtt{NC}[1;4]$, we apply Lemma 5.2.6 for $\alpha(x) := \neg u^T_\alpha(x)$, $\beta(y) := u^T_\alpha(y)$ and respectively clauses 3, 3, and $\neg u^T_\alpha(y) \vee u^T_\alpha(y)$ for $\mathtt{R}_1$, $\mathtt{R}_2$ and $\mathtt{R}_3$. Since these clauses are in N or redundant w.r.t. N, the inference $\mathtt{NC}[1;4]$ is redundant as well. □

Lemma 5.2.7 says that all that we need in order to avoid dangerous Negative Chaining inferences between clauses of forms $C \vee sTt$ and $\neg(xTy) \vee \alpha[x] \vee \beta[y]$, is to make a Transitive Closure inference with the second clause and resolve the first clause with its first two conclusions.

A saturation strategy for $\mathcal{GF}[\mathsf{TG}]$

In order to obtain a saturation-based decision procedure for $\mathcal{GF}[\mathsf{TG}]$, this time we need to take a special care about extended predicate symbols produced by the Transitive Closure rule (see Figure 5.4), since, in contrast to the Literal Projection rule (see Figure 4.1 on p.112), we are no longer guaranteed that only *finitely many* of these are produced in a saturation. It might be possible, that the Transitive Closure rule is repeatedly applied to clauses that contain new predicate symbols introduced by this rule, which results in an infinite expansion of the signature. Our intension is to show that this does *not* happen for $\mathcal{GF}[\mathsf{TG}]$.

We define a clause class $(\mathbf{G}^T)$ for $\mathcal{GF}[\mathsf{TG}]$ (see Table 5.2). This clause class is in somewhat similar to clause class $(\mathbf{G}^f)$ defined for the *guarded fragment with functional guards* $\mathcal{GF}[\mathsf{FG}]$ in subsection 4.3.3 (see Table 4.18). In particular, here we also separate clauses that contain *positive occurrences* of special, i.e., transitive atoms: $\mathtt{U}.1 - \mathtt{U}.3$. All remaining clauses may contain transitive atoms only *negatively.*

Table 5.2 A clause class for the guarded fragment with transitive guards

	Clause scheme		*Description*
$(\mathbf{G}^T)$:	1	$\hat{\beta}[\hat{c}]$	*a ground clause containing transitive predicate symbols only negatively;*
	2	$\neg\hat{a}[!\overline{x}] \vee \alpha[\overline{x}] \vee \hat{\beta}[!\hat{f}(\overline{x}), \overline{x}]$	*a guarded clause whose extended atoms introduced by the* Transitive Closure *rule contain functional symbols;*
	T	$\neg\{!\hat{T}\}[!x, !y] \vee \hat{\alpha}[x] \vee \hat{\alpha}[y]$	*an instance of the previous scheme where all literals containing different variables are transitive and occur negatively*
	U.1	$\neg p(x) \vee h_l^T(x)Tx$	*clauses that originate from positive occurrences of transitive guards; every Skolem function for these guards is labelled according to whether it has been introduced for its "left" variable, its "right" variable.*
	U.2	$\neg p(x) \vee xTh_r^T(x)$	
	U.3	$\neg p(x) \vee xTx$	
	U.4	$\hat{\beta}[h^T(x), x]$	*clauses with one variable that may contain a Skolem function introduced for a transitive guard*
	where $a := p \mid T;$ $l := p \mid \neg p \mid \neg T;$ $\alpha := \vee\{l\};$ $h^T := h_l^T \mid h_r^T;$ $q := p \mid u^S_{\hat{\alpha}};$ $b := a \mid u^S_{\hat{\alpha}};$ $k := l \mid u^S_{\hat{\alpha}} \mid \neg u^S_{\hat{\alpha}};$ $\beta := \vee\{k\};$		

In order to ensure that the Transitive Closure rule does not produce infinitely many *extended predicate symbols*, we have introduced additional *parameters* q, b,

k and β, respectively for non-special atoms, atoms, literals and clauses that might contain extended predicate symbols resulted from this rule. The important property which guarantees that only *finitely* many of such symbols are produced, is that for clauses of form 2, the literals formed from such symbols must contain a functional term. This implies that clauses of form T (which are instances of form 2), *do not contain* such symbols.

We briefly summarise our saturation strategy and arguments which insure that the class $(\mathbf{G}^T)$ is closed under all inferences. The detailed case analysis can be found in Appendix D.2:

- *We apply resolution inferences with clauses of forms* 1 *and* 2 *as for the guarded fragment: if a clause contains a functional term, we resolve on its maximal literal. Otherwise we select a* non-special *guard, as long as there exists one. The remaining case is considered below.*
  Note that no Negative Chaining *inferences with clauses of form* 2 *are possible, since transitive atoms may occur positively only in clauses of form* U.1 – U.3*, and they either contain a functional term $h(x)$ which does not occur in clauses of form* 2*, or are of the form xTx, which could not be used in any chaining inference with transitive relations.*
  Note also that no Compositional Resolution *inferences with clauses of* $(\mathbf{G}^T)$ *are possible, since every clause contains at most one positive occurrence of transitive atoms.*

- *If a clause of form* 2 *contains neither functional terms nor non-special guards, i.e., it is of form $\neg\{{!}\hat{T}\}[{!}x, {!}y] \vee \hat{\alpha}[x, y]$, then its* maximal *literal either contains both variables x and y – in this case we resolve on this literal, or it contains* duplicate occurrences *of some variable – in this case we apply the* Literal Projection *rule, i.e., the strategy is similar as for the two-variable fragment (see subsection 4.1.2). Otherwise the clause should be of form* T.

- *For clauses of form* T *we face with the problem of* Negative Chaining *inferences. In case when there is* only one *transitive guard in such clause, we produce inferences by the* Transitive Closure *rule, which solves the problem according to Lemma 5.2.7.*
  Otherwise, when there are several *transitive guards in a clause, we apply the* Negative Hyper-Chaining *rule on them (see Figure 3.5 on p.88). This rule could be applied only in one case – for a clause of form $\neg(\boldsymbol{xT\underline{y}})^\sharp \vee \neg(\boldsymbol{yT\underline{x}})^\sharp \vee \alpha[x, y]$. In all other cases no* Negative Hyper-Chaining *inference is possible since the functional terms in clauses* U.1 *and* U.2 uniquely determine *their transitive predicate symbol and the side (left or right) in which they occur. Moreover, for this remaining case the variables x and y* will be unified *because of the condition (vi) of the* Negative Hyper-Chaining *rule.*

Complexity

The case analysis of possible inferences between clauses from $(\mathbf{G}^T)$ (which can be found in Appendix D.2), proves that this clause class is closed under non-redundant chaining inferences, provided that certain Transitive Closure inferences are drawn. This gives a decision procedure for $\mathcal{GF}[\mathsf{TG}]$. It is possible to show that the complexity of this decision procedure is 2EXPTIME in the worst case. All computations are similar to the case with the guarded fragment, except that now we may have *exponentially many unary* predicate symbols resulted from the Transitive Closure rule. The reason is that the extended predicate symbols u^T_α produced by this rule are indexed with a *subclause* $\alpha[x]$, and there might be exponentially many of those. Nonetheless, the overall complexity remains the same, since the number of guarded clauses is doubly exponential in the *number of different variables* in clauses, and only *singly exponential* in the size of the vocabulary (see estimation (C.3) on p.240).

Unfortunately, since vocabulary now can be exponentially large, the complexity remains *doubly exponential* even for the bounded variable version $\mathcal{GF}^k[\mathsf{TG}]$ of $\mathcal{GF}[\mathsf{TG}]$. In fact, $\mathcal{GF}^k[\mathsf{TG}]$ and even $\mathcal{GF}^2[\mathsf{TG}]$ is 2EXPTIME-hard, as has been shown by Kieronski [2003]. So, our procedure is *theoretically optimal* in this case either. Moreover, for many subclasses of $\mathcal{GF}^2[\mathsf{TG}]$, in particular for the modal fragment $\mathcal{FO}(\mathrm{MF})$ (3.6) (see subsection 3.2.1) with transitivity, our procedure is in EXPTIME. Indeed, it is possible to show that all clauses of form 2 with a transitive guard, that are obtained from such formulas will be already of form $\mathtt{T}$. Hence, the Transitive Closure rule is applied only to the input clauses and produces only *linear* number of extended predicate symbols. This indicates the "pay as you go" behavior of our decision procedure: is not only optimal for the full $\mathcal{GF}[\mathsf{TG}]$, but also for its simpler subfragments.

Theorem 5.2.8. *There is a saturation-based decision procedure for the guarded fragment with transitive guards* $\mathcal{GF}[\mathsf{TG}]$ *without equality, which can be implemented in 2EXPTIME. This procedure decides the modal fragment* $\mathcal{FO}(\mathrm{MF})$ *with transitive relations in EXPTIME.*

5.2.2 Deciding the Guarded Fragment with Compositional Guards

In this section we extend the procedure for $\mathcal{GF}[\mathsf{TG}]$ described in the previous section, to a wider class of compositional theories defined by *associative compositional axioms*. Such extension seems to be not difficult, since the chaining calculus for compositional relations is a straightforward generalisation of the one for transitive relations.

Let $\boldsymbol{\mathcal{C}}$ be a *compositional theory* consisting of *associative compositional axioms* of form $S \circ T \subseteq H$ (recall terminology from 3.5.4). We define a *guarded fragment with compositional guards* $\mathcal{GF}[\mathsf{CG}]$ (over $\boldsymbol{\mathcal{C}}$) to be the set of guarded formulas in which every special (compositional) binary atom occurs only in guards.

Redundancy of Negative Chaining inferences

First of all, we extend the Transitive Closure rule to compositional relations in order to insure redundancy for Negative Chaining inferences. This extension is given in Figure 5.5. Note that conclusions of this rule are produced for *all* (intermediate) $S \circ T \subseteq H$ such that $H \circ V \subseteq U$ or $H = U$. This is needed to insure not only redundancy of Negative Chaining inferences with the premise of this rule, but also with the conclusions of this rule. In other words, the premise and *all* conclusions must be properly “closed”. Below we give more details on this.

Figure 5.5 The Compositional Closure rule

Compositional Closure

$$\mathtt{CC}: \frac{\neg\boldsymbol{(xUy)}^{\sharp} \vee \alpha[x] \vee \beta[y]}{\begin{array}{l} \neg(xSy) \vee \alpha[x] \vee u^{S}_{\alpha}(y) \\ \neg(xTy) \vee \neg u^{S}_{\alpha}(x) \vee u^{H}_{\alpha}(y) \\ \qquad\qquad\;\; \neg u^{U}_{\alpha}(y) \vee \beta[y] \end{array}}$$

$\left[\begin{array}{l} \textit{where (i) } S \circ T \subseteq H\textit{, (ii) } H \circ V \subseteq U \textit{ for some } V\textit{, or } H = U\textit{,} \\ \textit{and (iii) } u^{S}_{\alpha}\textit{, } u^{H}_{\alpha} \textit{ and } u^{U}_{\alpha} \textit{ are extended unary predicate symbols} \\ \textit{introduced for } \alpha \textit{ and, respectively for } S\textit{, } H \textit{ and } U\textit{.} \end{array}\right]$

The Compositional Closure rule is a proper *generalisation* of the Transitive Closure rule, and can be proven to be sound in the same way.

Recall from 3.5.4 that in order to demonstrate redundancy of clauses and inferences in the ordered chaining calculus for *compositional* relations, we need to additionally supply a *redundancy ordering* $\succ\!\!-$ that is compatible with our main ordering $\succ\!\!-$ according to Definition 3.5.7. In fact, the (*partial*) ordering induced by the weight function (5.13) admits all these required properties. Recall, that it also fulfils all the properties of admissible ordering, except for totality, it is *simple*, *CAP* and *CASP*. So, from now on we assume such *redundancy ordering* $\succ\!\!-$ to be given.

Now we generalise Lemma 5.2.6 to associative compositional theories as follows:

Lemma 5.2.9. *Let N be a clause set such that the following clauses are contained in N or are redundant w.r.t. N:*

$\mathtt{R}_0$. $\neg\boldsymbol{(xHy)}^{\sharp} \vee \alpha[x] \vee \beta[y]$; $\mathtt{R}_2$. $\neg\boldsymbol{(xTy)} \vee \neg u_1(x) \vee u_2(y)$;
$\mathtt{R}_1$. $\neg\boldsymbol{(xSy)} \vee \alpha[x] \vee u_1(y)$; $\mathtt{R}_3$. $\neg u_2(y) \vee \beta[y]$;

for some $u_1(x)$, $u_2(x)$ *and* $S \circ T \subseteq H$*. Then in addition:*

(***a***) *If* N *contains the clauses*

$\mathtt{1}_a.\ C \vee \boldsymbol{sSt};\qquad \mathtt{OR}[1_a;\mathtt{R}_1]:\ \mathtt{2}_a.\ C \vee \alpha[s] \vee u_1(t),$

then the following Negative Chaining *inference is redundant w.r.t.* N*:*

$\mathtt{NC}[1_a;\mathtt{R}_0]:\ \mathtt{3}_a.\ C \vee \neg(tTy) \vee \alpha[s] \vee \beta[y];$

(***b***) *If* N *contains the clauses*

$\mathtt{1}_b.\ C \vee \boldsymbol{tTs};\qquad \mathtt{OR}[1_b;\mathtt{R}_2]:\ \mathtt{2}_b.\ C \vee \neg u_1(t) \vee u_2(s)$

then the following Negative Chaining *inference is redundant w.r.t.* N*:*

$\mathtt{NC}[1_b;\mathtt{R}_0]:\ \mathtt{3}_b.\ C \vee \neg(xSt) \vee \alpha[x] \vee \beta[s].$

Proof. Can be found in Appendix D.1. ⧉

Lemma 5.2.10. *Let* N *be a clause set containing clauses:*

$1.\ C \vee \boldsymbol{sRt},\qquad 2.\ \boldsymbol{\neg(xUy)}^{\sharp} \vee \alpha[x] \vee \beta[y];$

together with all conclusions of the Compositional Closure *rule from* 2 *and their resolvents with 1:*

$$\begin{array}{lll} \mathtt{TC}[2]: & \mathtt{3}.\ \boldsymbol{\neg(xSy)} \vee \alpha[x] \vee u^{S}_{\alpha}(y); & \mathtt{OR}[1;3]{:}\, C \vee \alpha[s] \vee u^{S}_{\alpha}(t) \quad \text{if } R = S; \\ & \mathtt{4}.\ \boldsymbol{\neg(xTy)} \vee \neg u^{S}_{\alpha}(x) \vee u^{H}_{\alpha}(y); & \mathtt{OR}[1;4]{:}\, C \vee \neg u^{S}_{\alpha}(s) \vee u^{H}_{\alpha}(t) \quad \text{if } R = T; \\ & \mathtt{5}.\ \neg u^{U}_{\alpha}(y) \vee \beta[y]; & \end{array}$$

Then all Negative Chaining *inferences between clause* 1 *and clauses* 2*,* 3 *and* 4 *are redundant.*

Proof. Can be found in Appendix D.1. ⧉

From Lemma 5.2.9 and Lemma 5.2.10 we see that not *all* conclusions of the Compositional Closure rule might be used for proving redundancy of Negative Chaining inferences. It is possible to make this rule more efficient (especially for large compositional theories over *distance* or *path relations*) by producing only those conclusions that are *needed for proving redundancy.* This optimised variant of the Compositional Closure rule is given in Figure 5.6

Redundancy of Negative Hyper-Chaining inferences

In our saturation-based decision procedure for $\mathcal{GF}[\mathsf{TG}]$, we have used the Negative Hyper-Chaining rule for clauses containing several transitive guards. In most cases such inferences were not possible because of the syntactical restrictions on positive occurrences of transitive atoms. This helped to avoid Negative Chaining inferences with such clauses.

Figure 5.6 An optimised variant of the Compositional Closure rule

Conditional Compositional Closure

$$\texttt{CCC}: \frac{C \vee \boldsymbol{sSt} \quad \boldsymbol{\neg(xHy)} \vee \alpha[x] \vee \beta[y]}{\begin{array}{r} \neg(xSy) \vee \alpha[x] \vee u^{S}_{\alpha}(y) \\ \neg(xTy) \vee \neg u^{S}_{\alpha}(x) \vee u^{H}_{\alpha}(y) \\ \neg u^{H}_{\alpha}(y) \vee \beta[y] \end{array}} \qquad \frac{C \vee \boldsymbol{sSt} \quad \boldsymbol{\neg(xHy)}^{\star} \vee \neg u^{U}_{\alpha}(x) \vee u^{W}_{\alpha}(y)}{\begin{array}{r} \neg(xSy) \vee \neg u^{U}_{\alpha}(x) \vee u^{V}_{\alpha}(y) \\ \neg(xTy) \vee \neg u^{V}_{\alpha}(x) \vee u^{W}_{\alpha}(y) \end{array}}$$

[*where (i)* $C \not\succeq sSt$; *(ii)* $t \not\succeq s$; *(iii)* $S \circ T \subseteq H$; *(iv)* u^{S}_{α} *and* u^{H}_{α} *are extended unary predicate symbols.*] [*where (i)* $C \not\succeq tTs$; *(ii)* $t \not\succeq s$; *(iii)* $S \circ T \subseteq H$; $U \circ H \subseteq W$; $U \circ S \subseteq V$ *(iv)* u^{U}_{α}, u^{V}_{α} *and* u^{W}_{α} *are extended unary predicate symbols.*]

$$\frac{C \vee \boldsymbol{tTs} \quad \boldsymbol{\neg(xHy)} \vee \alpha[x] \vee \beta[y]}{\begin{array}{r} \neg(xSy) \vee \alpha[x] \vee u^{S}_{\alpha}(y) \\ \neg(xTy) \vee \neg u^{S}_{\alpha}(x) \vee u^{H}_{\alpha}(y) \\ \neg u^{H}_{\alpha}(y) \vee \beta[y] \end{array}} \qquad \frac{C \vee \boldsymbol{tTs} \quad \boldsymbol{\neg(xHy)}^{\star} \vee \neg u^{U}_{\alpha}(x) \vee u^{W}_{\alpha}(y)}{\begin{array}{r} \neg(xSy) \vee \neg u^{U}_{\alpha}(x) \vee u^{V}_{\alpha}(y) \\ \neg(xTy) \vee \neg u^{V}_{\alpha}(x) \vee u^{W}_{\alpha}(y) \end{array}}$$

[*where (i)* $C \not\succeq tTs$; *(ii)* $t \not\succeq s$; *(iii)* $S \circ T \subseteq H$; *(iv)* u^{S}_{α} *and* u^{H}_{α} *are extended unary predicate symbols.*] [*where (i)* $C \not\succeq tTs$; *(ii)* $t \not\succeq s$; *(iii)* $S \circ T \subseteq H$; $U \circ H \subseteq W$; $U \circ S \subseteq V$ *(iv)* u^{U}_{α}, u^{V}_{α} *and* u^{W}_{α} *are extended unary predicate symbols.*]

It turns out, that many Negative Hyper-Chaining inferences can be made redundant using an extension of the Compositional Closure rule given in Figure 5.7. According to this rule, we introduce extended unary predicate symbols $u^{\tilde{S_1}..\tilde{S_n}}_{\alpha}$ for all special guards $\neg(x\tilde{S}_1y), \ldots, \neg(x\tilde{S}_ny)$ of a clause (recall that $\tilde{S}$ denotes S or its inverse $S^{\smile}$). Semantically, $u^{\tilde{S_1}..\tilde{S_n}}_{\alpha}(y)$ represents the set of all elements that are reachable using *all* these relations from some element x such that $\alpha(x)$ does *not* hold, i.e., we should have $x\tilde{S}_1y, \ldots, x\tilde{S}_ny$. It is possible to show that Multi-Compositional Closure is a sound inference rule and that the following analog of Lemma 5.2.10 holds:

Figure 5.7 An extension of the Compositional Closure rule to several compositional relations

Multi-Compositional Closure

$$\texttt{MCC}: \frac{\boldsymbol{\neg(x\tilde{U}_1y)}^{\sharp} \vee \cdots \vee \boldsymbol{\neg(x\tilde{U}_ny)}^{\sharp} \vee \alpha[x] \vee \beta[y]}{\begin{array}{r} \neg(x\tilde{S}_1y) \vee \cdots \vee \neg(x\tilde{S}_ny) \vee \alpha[x] \vee u^{\tilde{S_1}..\tilde{S_n}}_{\alpha}(y) \\ \neg(x\tilde{T}_1y) \vee \cdots \vee \neg(x\tilde{T}_ny) \vee \neg u^{\tilde{S_1}..\tilde{S_n}}_{\alpha}(x) \vee u^{\tilde{H_1}..\tilde{H_n}}_{\alpha}(y) \\ \neg u^{\tilde{U_1}..\tilde{U_n}}_{\alpha}(y) \vee \beta[y] \end{array}}$$

[*where (i)* $\tilde{S}_i \circ \tilde{T}_i \subseteq \tilde{H}_i$, *(ii)* $\tilde{H}_i \circ \tilde{V}_i \subseteq \tilde{U}_i$, *for some* V_i, *or* $\tilde{V}_i = \tilde{H}_i$, $1 \leq i \leq n$, *and (iii)* $u^{\tilde{S_1}..\tilde{S_n}}_{\alpha}$, $u^{\tilde{H_1}..\tilde{H_n}}_{\alpha}$ *and* $u^{\tilde{U_1}..\tilde{U_n}}_{\alpha}$ *are extended unary predicate symbols.*]

Lemma 5.2.11. *Let N be a clause set containing clauses:*

1_i. $C_i \vee \boldsymbol{s\tilde{R}_i t}$, $1 \leq i \leq n$ 2. $\neg\boldsymbol{(x\tilde{U}_1 y)}^\sharp \vee \cdots \vee \neg\boldsymbol{(x\tilde{U}_n y)}^\sharp \vee \alpha[x] \vee \beta[y]$;

and all conclusions of Multi-Compositional Closure *from* 2 *and their hyper-resolvents with 1:*

$$\begin{aligned}
\mathtt{MCC}[\mathrm{R}_0]:\ & 3.\ \neg\boldsymbol{(x\tilde{S}_1 y)}^\sharp \vee \cdots \vee \neg\boldsymbol{(x\tilde{S}_n y)}^\sharp \vee \alpha[x] \vee u_\alpha^{\tilde{S_1}..\tilde{S_n}}(y);\\
& 4.\ \neg\boldsymbol{(x\tilde{T}_1 y)}^\sharp \vee \cdots \vee \neg\boldsymbol{(x\tilde{T}_n y)}^\sharp \vee \neg u_\alpha^{\tilde{S_1}..\tilde{S_n}}(x) \vee u_\alpha^{\tilde{H_1}..\tilde{H_n}}(y);\\
& 5.\ \neg u_\alpha^{\tilde{U_1}..\tilde{U_n}}(y) \vee \beta[y];
\end{aligned}$$

$$\begin{aligned}
&\mathtt{HR}[1_1,..,1_n;3]: C_1 \vee \cdots \vee C_n \vee \alpha[s] \vee u_\alpha^{\tilde{S_1}..\tilde{S_n}}(t) && \textit{if } \tilde{R}_i = \tilde{S}_i,\ 1 \leq i \leq n;\\
&\mathtt{HR}[1_1,..,1_n;4]: C_1 \vee \cdots \vee C_n \vee \neg u_\alpha^{\tilde{S_1}..\tilde{S_n}}(s) \vee u_\alpha^{\tilde{H_1}..\tilde{H_n}}(t) && \textit{if } \tilde{R}_i = \tilde{T}_i,\ 1 \leq i \leq n;
\end{aligned}$$

Then all Negative Hyper-Chaining *inferences between clause* 1_i *and clauses* 2, 3 *and* 4 *on variable x or on variable y are redundant.*

Note that terms s and t must be the same in *all* atoms $s\tilde{R}_i t$ of clauses 1_i in order to make inference redundant. In this case a proof for Lemma 5.2.11 goes in the same way as for Lemma 5.2.10: we just need to duplicate correspondent literals and clauses. Note also, that redundancy is established when chaining is performed with either variable x or variable y. The lemma does not say anything when we have a "mixed" chaining into both variables. However, in the letter case, variables x and y must be unified because of condition (vi) of the Negative Hyper-Chaining rule (see Figure 3.5 on p.88). Hence such inferences are not dangerous.

A saturation strategy for $\mathcal{GF}[\mathsf{CG}]$

Using of the Multi-Compositional Closure rule allows us to simplify our saturation strategy for $\mathcal{GF}[\mathsf{CG}]$. We might no longer use syntactical restrictions on the positive occurrences of special atoms, that we had for clauses U.1 – U.3 from clause class $(\mathbf{G}^T)$ (see Table 5.2), to block dangerous Negative Hyper-Chaining inferences. Instead of this, we may use Multi-Compositional Closure for making them redundant according to Lemma 5.2.11.

Taking these considerations into account, we merge clause schemes U.1 – U.3 into one clause scheme U: see Table 5.3. Now a Negative Hyper-Chaining inference is possible only between clauses of form $C_1 \vee \boldsymbol{x\tilde{S}_1 \underline{h(x)}}^\star, \ldots, C_n \vee \boldsymbol{x\tilde{S}_n \underline{h(x)}}^\star$ from U and a clause of form S. The main point is that term $h(x)$ must be *the same* in all these clauses since they are unified in the inference. This makes it possible to apply Lemma 5.2.11.

Apart from that, our saturation strategy does not differ much from the one for $(\mathbf{G}^T)$. The case analysis of all inferences between clauses from $(\mathbf{G}^S)$ can be found in Appendix D.3.

Table 5.3 A clause class for the guarded fragment with compositional guards

	Clause scheme	***Description***
$(\mathbf{G}^S)$:	1 $\hat{\beta}[\hat{c}]$	*a ground clause containing compositional predicate symbols only negatively;*
	2 $\neg\hat{a}[!\overline{x}] \vee \alpha[\overline{x}] \vee \hat{\beta}[!\hat{f}(\overline{x}), \overline{x}]$	*a guarded clause whose extended atoms contain functional symbols;*
	`S` $\neg\{!\hat{S}\}[!x, !y] \vee \alpha[x] \vee \alpha[y]$	*an instance of the previous scheme where all literals containing different variables are compositional and occur negatively*
	`U` $\{\neg\hat{S}, \hat{S}, \hat{k}\}[h(x), x]$	*clauses with one variable that may contain a Skolem function introduced for a compositional guard*
	where $a := p \,\vert\, S$; $l := p \,\vert\, \neg p \,\vert\, \neg S$; $\alpha := \vee\{l\}$; $q := p \,\vert\, u^S_{\hat{\alpha}}$; $b := a \,\vert\, u^S_{\hat{\alpha}}$; $k := l \,\vert\, u^S_{\hat{\alpha}} \,\vert\, \neg u^S_{\hat{\alpha}}$; $\beta := \vee\{k\}$;	

Theorem 5.2.12. *There is a saturation-based decision procedure for the guarded fragment with transitive guards* $\mathcal{GF}[\mathsf{CG}]$ *without equality, which can be implemented in 2EXPTIME.*

$\mathcal{GF}$ with conjunctions of compositional guards

It turns out that the extended clause class $(\mathbf{G}^S)$ makes it possible to obtain decidability results for a *larger* fragment that $\mathcal{GF}[\mathsf{CG}]$. Recall that we have extended a set of clauses of form `U`, and, in particular, they can contain *several* special atoms positively, and functions $h(x)$ are no longer indexed with special atoms.

Consider a fragment $\mathcal{GF}[\wedge\mathsf{CG}]$, which is defined as $\mathcal{GF}[\mathsf{CG}]$, except that we also allow guards of form $xS_1y \wedge \cdots \wedge xS_ny$, where S_1,.., S_n are special predicate symbols, $n \geq 1$. We call such extension of the guarded fragment by the *guarded fragment with conjunction of compositional guards*. It is possible to show, that the standard **CNF**-transformation for the formulas from this fragment produces clauses from $(\mathbf{G}^S)$. Indeed, the only cases that are affected in a translation are listed in Table 4.18 on p.140, where we should now replace atom $r(x, y)$ with conjunction of such atoms. This gives us either guarded clauses (6.1) – (6.3) containing possibly several special guards, or other clauses containing at most one positive occurrence of a special atom, which might share the same Skolem term $h(x)$. All these clauses belong to $(\mathbf{G}^S)$, which implies:

Theorem 5.2.13. *There is a saturation-based decision procedure that decides* $\mathcal{GF}[\wedge\mathsf{CG}]$ *without equality, which can be implemented in 2EXPTIME.*

5.2.3 Undecidability of the Guarded Fragment over Relational Algebras

In the previous section we have demonstrated that when restricting the set of compositional axioms to “regular” ones, it is possible to obtain decidable fragments that subsume many expressive modal and description logics. Now we are wondering whether this approach can be extended to more general classes of compositional axioms of form:

$$S \circ T \subseteq H_1 \cup \cdots \cup H_n \tag{5.14}$$

Unfortunately, axioms of form 5.14 might lead to *undecidable* modal and description logics even if they admit many good properties such as *associativity*. In this section we give an example of a simple relational algebra (see Definition 5.1.1) such that a modal logic defined over relations of this algebra is *undecidable*.

Consider the following *distance relations*: $x\,\mathtt{D}_{\le 1}\,y$, $x\,\mathtt{D}_{(1,2]}\,y$ and $x\,\mathtt{D}_{>2}\,y$, which read as "the distance between x and y is smaller or equal than 1", respectively, "greater than one but smaller or equal than two", or "greater than two". These relations are mutually disjoint, symmetric: $\mathtt{D}_{\le 1} \equiv \mathtt{D}_{\le 1}{}^{\smile}$, $\mathtt{D}_{(1,2]} \equiv \mathtt{D}_{(1,2]}{}^{\smile}$ $\mathtt{D}_{>2} \equiv \mathtt{D}_{>2}{}^{\smile}$, and admit the following compositional axioms of form (5.14):

$\circ$	$\mathtt{D}_{\le 1}$	$\mathtt{D}_{(1,2]}$	$\mathtt{D}_{>2}$
$\mathtt{D}_{\le 1}$	$\mathtt{D}_{\le 1} \cup \mathtt{D}_{(1,2]}$	$\mathtt{D}_{\le 1} \cup \mathtt{D}_{(1,2]} \cup \mathtt{D}_{>2}$	$\mathtt{D}_{(1,2]} \cup \mathtt{D}_{>2}$
$\mathtt{D}_{(1,2]}$	$\mathtt{D}_{\le 1} \cup \mathtt{D}_{(1,2]} \cup \mathtt{D}_{>2}$	$\mathtt{D}_{\le 1} \cup \mathtt{D}_{(1,2]} \cup \mathtt{D}_{>2}$	$\mathtt{D}_{\le 1} \cup \mathtt{D}_{(1,2]} \cup \mathtt{D}_{>2}$
$\mathtt{D}_{>2}$	$\mathtt{D}_{(1,2]} \cup \mathtt{D}_{>2}$	$\mathtt{D}_{\le 1} \cup \mathtt{D}_{(1,2]} \cup \mathtt{D}_{>2}$	$\mathtt{D}_{\le 1} \cup \mathtt{D}_{(1,2]} \cup \mathtt{D}_{>2}$

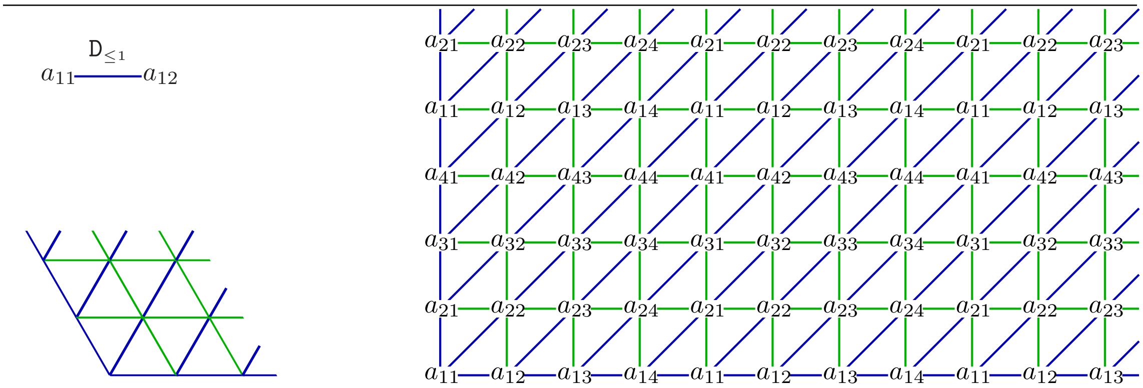
(5.15)

It is possible to extend the set of these relations to a *relational algebra* satisfying all conditions (5.5) from Definition 5.1.1: for the identity Id we take an additional relation $x\mathtt{D}_{=0}\,y$ and then consider all possible boolean combinations of these relations for A. In particular, relations $\mathtt{D}_{\le 1}$, $\mathtt{D}_{(1,2]}$ and $\mathtt{D}_{>2}$ satisfy the *associativity* property and the *triangle axiom*.

We demonstrate that many “interesting” extensions of description logics and the guarded fragment with relations $\mathtt{D}_{\le 1}$, $\mathtt{D}_{(1,2]}$ and $\mathtt{D}_{>2}$ are *undecidable*. Our proof is, as usual, by a reduction from domino problems, and is in somewhat similar to undecidability proof given in [Kutz et al., 2003], which shows that certain extensions of modal logics with distance relations of form $\mathtt{D}_{(0;n]}$ might lead to undecidability. However our construction is much simple.

Using relations $\mathtt{D}_{\le 1}$ and $\mathtt{D}_{(1,2]}$, we encode a grid structure shown in Figure 5.8. Here all indicated edges (including diagonal ones) have the length ≤ 1 and distances between all different nodes that are not connected with an edge is > 1. Such situation is “physically” possible, if we stretch the angle between the initial axes to 120° (see the small illustration in the left corner).

Figure 5.8 Undecidability of $\mathcal{GF}^2$ with guards over a relational algebra



$\mathtt{D}_{\leq 1}$
a_{11} a_{12}
a_{21} a_{22} a_{23} a_{24} a_{21} a_{22} a_{23} a_{24} a_{21} a_{22} a_{23}
a_{11} a_{12} a_{13} a_{14} a_{11} a_{12} a_{13} a_{14} a_{11} a_{12} a_{13}
a_{41} a_{42} a_{43} a_{44} a_{41} a_{42} a_{43} a_{44} a_{41} a_{42} a_{43}
a_{31} a_{32} a_{33} a_{34} a_{31} a_{32} a_{33} a_{34} a_{31} a_{32} a_{33}
a_{21} a_{22} a_{23} a_{24} a_{21} a_{22} a_{23} a_{24} a_{21} a_{22} a_{23}
a_{11} a_{12} a_{13} a_{14} a_{11} a_{12} a_{13} a_{14} a_{11} a_{12} a_{13}

$$
\begin{array}{ll}
\mathsf{GRID} \;:=\; \exists x.[a_{11}(x) \wedge \mathtt{I}(x) \wedge \mathtt{J}(x)] \wedge & \text{- \textit{creates the origin of a grid}} \\
\wedge\, \forall x.[\mathtt{I}(x) \rightarrow \bigwedge\limits_{i' = i+1 \bmod 4} (a_{i1}(x) \rightarrow \exists y.[x\, \mathtt{D}_{\leq 1}\, y \wedge a_{i'1}(y)])\ ] \wedge & \text{- \textit{creates the initial horizontal}} \\
\wedge\, \forall x.[\mathtt{J}(x) \rightarrow \bigwedge\limits_{j' = j+1 \bmod 4} (a_{1j}(x) \rightarrow \exists y.[x\, \mathtt{D}_{\leq 1}\, y \wedge a_{1j'}(y)])\ ] \wedge & \text{- \textit{and vertical axes}} \\
\wedge \bigwedge\limits_{\substack{i' = i+1 \bmod 4 \\ j' = j+1 \bmod 4}} \forall x.(a_{ij}(x) \rightarrow \exists y.[x\, \mathtt{D}_{\leq 1}\, y \wedge a_{i'j'}(y)]) \wedge & \text{- \textit{launches the diagonals}} \\
\wedge\, \forall xy.(\mathtt{D}_{(1,2]}(x,y) \rightarrow \bigwedge\limits_{j' = j+1 \bmod 4} \neg[a_{ij}(x) \wedge a_{ij'}(x)] \;\wedge & \text{- \textit{forbids intermediate}} \\
\qquad \wedge \bigwedge\limits_{i' = i+1 \bmod 4} \neg[a_{ij}(x) \wedge a_{i'j}(x)]\ ) \wedge & \text{- \qquad \textit{distances for close nodes}} \\
\wedge\, \forall xy.[x\, \mathtt{D}_{\leq 1}\, y \rightarrow (\bigvee\limits_{j' = j+1 \bmod 4} [a_{ij}(x) \wedge a_{ij'}(x)]\ ) \rightarrow H(x,y)] \wedge & \\
\wedge\, \forall xy.[x\, \mathtt{D}_{\leq 1}\, y \rightarrow (\bigvee\limits_{i' = i+1 \bmod 4} [a_{ij}(x) \wedge a_{i'j}(x)]\ ) \rightarrow V(x,y)] & \text{- \textit{entails edges of a grid}}
\end{array}
$$

In order to enforce a grid structure from Figure 5.8, we first create a “skeleton” of a grid consisting of the initial axes and diagonal lines, and then make other edges by forbidding the distances between close nodes to be greater than 1. This can be done using formula GRID in Figure 5.8. The conjunct starting with $\forall xy.(x\, \mathtt{D}_{(1,2]}\, y \rightarrow \cdots$ is the most interesting in this formula. It expresses that the distance between nodes labelled with *consequent* labels could not be greater than 1 and smaller then 2. In other words, all such pairs of nodes must be either within the distance ≤ 1 from each other, or the distance between them must be > 2. It can be shown by induction, that the last is not possible for the pairs of neighbouring nodes of the grid: this follows from compositional axioms (5.15). Indeed, the distance between first two nodes on the second line (labelled respectively with a_{21} and a_{22}) could not be greater than 2 because of the triangular inequalities, since the lengths of the diagonal and the edge (a_{11}, a_{21}) are not greater than 1. Hence this distance must be ≤ 1 because of this conjunct of GRID. Similarly, such constrains are propagated to all edges. It is important that "distant" nodes with consequent labels will not be connected with an edge, because it is possible for them to be on the distance greater than 2.

It can be shown that formula $\mathsf{GRID} \wedge \mathsf{TILING}$ can be expressed in $\mathcal{ALC}$ using only roles $\mathtt{D}_{\leq 1}$, $\mathtt{D}_{(1,2]}$ and $\mathtt{D}_{>2}$ (the last is used only for internalisation of TBox-es in a similar way as it has been done in the end of subsection 5.1.3). Hence this reduction shows that subsumption and satisfiability in $\mathcal{ALC}_{\mathcal{RA}}$ – $\mathcal{ALC}$ whose roles are interpreted over a relational algebra $\mathcal{RA}$ – is *undecidable* in general. In particular, the guarded fragment with guards from a relational algebra $\mathcal{GF}[\mathsf{RAG}]$ is undecidable as well.

Theorem 5.2.14. *There exists a relational algebra $\mathcal{RA}$ such that $\mathcal{GF}^2[\mathsf{RAG}]$ is a conservative reduction class.*

An open question is, do these undecidability results extend for the particular relational algebras of *spatial relations* and *time intervals*?

5.3 Extensions with Equality

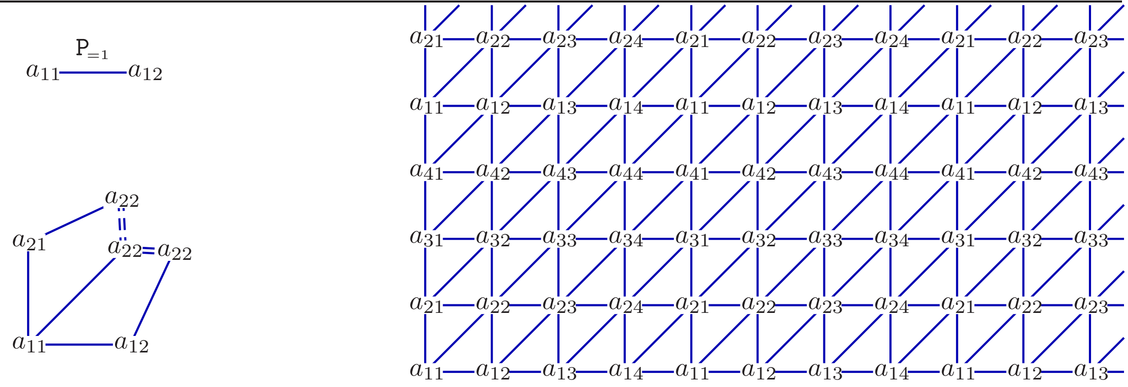
In this section we try to extend the results obtained in the previous section to the case with equality. Unfortunately, as will be seen in a moment, decidability for most extensions of the guarded fragment with compositional axioms is lost when equality is allowed. In particular, we show that, neither the guarded fragment with compositional guards, nor the guarded fragment with conjunctions of transitive guards remain decidable with equality.

The only fragment that remains decidable with equality is the *guarded fragment with transitive guards* $\mathcal{GF}_{\simeq}[\mathsf{TG}]$. This has been first demonstrated in [Szwast & Tendera, 2001]. We show how our saturation-based procedure can be extended to this fragment using an analog of the $\mathsf{Transitive\ Closure}$ rule which deals with a case when equational atoms come to play.

5.3.1 Undecidability for Associative Compositional Axioms

In this section we demonstrate undecidability for the *guarded fragment with compositional guards and equality* $\mathcal{GF}_{\simeq}[\mathsf{CG}]$. For proving this result, we enforce the same grid structure from Figure 5.8, used in subsection 5.2.3 for showing undecidability of $\mathcal{GF}[\mathsf{RAG}]$, but now using *path relations* instead of *distance relations*.

Figure 5.9 Undecidability of $\mathcal{GF}^2_{\simeq}$ with compositional guards


$\mathtt{P}_{=1}$
a_{11} a_{12}
a_{22}
a_{21} a_{22}=a_{22}
a_{11} a_{12}
a_{21} a_{22} a_{23} a_{24} a_{21} a_{22} a_{23} a_{24} a_{21} a_{22} a_{23}
a_{11} a_{12} a_{13} a_{14} a_{11} a_{12} a_{13} a_{14} a_{11} a_{12} a_{13}
a_{41} a_{42} a_{43} a_{44} a_{41} a_{42} a_{43} a_{44} a_{41} a_{42} a_{43}
a_{31} a_{32} a_{33} a_{34} a_{31} a_{32} a_{33} a_{34} a_{31} a_{32} a_{33}
a_{21} a_{22} a_{23} a_{24} a_{21} a_{22} a_{23} a_{24} a_{21} a_{22} a_{23}
a_{11} a_{12} a_{13} a_{14} a_{11} a_{12} a_{13} a_{14} a_{11} a_{12} a_{13}

$$\begin{array}{lll}
\mathsf{GRID} \;:=\; \exists x.a_{11}(x) \wedge & & \textit{- creates the origin of a grid}\\
\wedge \bigwedge_{j'=j+1 \bmod 4} \forall x.(a_{ij}(x) \rightarrow \exists y.[x\,\mathtt{P}_{=1}\,y \wedge H(x,y) \wedge a_{ij'}(y)]) \wedge & & \textit{- creates all horizontal edges}\\
\wedge \bigwedge_{i'=i+1 \bmod 4} \forall x.(a_{ij}(x) \rightarrow \exists y.[x\,\mathtt{P}_{=1}\,y \wedge V(x,y) \wedge a_{i'j}(y)]) \wedge & & \textit{- creates all vertical edges}\\
\wedge \bigwedge_{\substack{i'=i+1 \bmod 4\\ j'=j+1 \bmod 4}} \forall x.(a_{ij}(x) \rightarrow \exists y.[y\,\mathtt{P}_{=1}\,x \wedge a_{i'j'}(y)]) \wedge & & \textit{- creates all diagonal edges}\\
\wedge \forall xy.[x\,\mathtt{P}_{=3}\,y \rightarrow (\bigvee_{1\le i,j\le 4} [a_{ij}(x) \wedge a_{ij}(y)] \;) \rightarrow \boldsymbol{x \simeq y}] & & \textit{- ``glues'' close nodes}\\
 & & \textit{- with the same label}
\end{array}$$

Consider the following *path relations*: $x\,\mathtt{P}_{=1}\,y$, $x\,\mathtt{P}_{=2}\,y$ and $x\,\mathtt{P}_{=3}\,y$, which describe that there exists a path between x and y of the length respectively 1, 2 and 3. These relations are symmetric: $\mathtt{P}_{=1} \equiv \mathtt{P}_{=1}{}^{\smile}$, $\mathtt{P}_{=2} \equiv \mathtt{P}_{=2}{}^{\smile}$, $\mathtt{P}_{=3} \equiv \mathtt{P}_{=3}{}^{\smile}$ and admit the following compositional axioms:

$$\mathtt{P}_{=1} \circ \mathtt{P}_{=1} \subseteq \mathtt{P}_{=2} \qquad \mathtt{P}_{=1} \circ \mathtt{P}_{=2} \subseteq \mathtt{P}_{=3} \qquad \mathtt{P}_{=2} \circ \mathtt{P}_{=1} \subseteq \mathtt{P}_{=3} \tag{5.16}$$

We demonstrate that these axioms and equality suffice for encoding a grid structure given in Figure 5.9. The idea is, to create first all required edges and then to "glue together" nodes with the same labels that are reachable from each other via three edges (see the illustration in the left corner of Figure 5.9). This can be expressed with the formula GRID given in Figure 5.9. The essential part of this formula is the last conjunct which expresses how the nodes must be "glued" together using an *equational* atom $x \simeq y$. In contrast to special atoms $x\,\mathtt{P}_{=1}\,y$, $x\,\mathtt{P}_{=2}\,y$ and $x\,\mathtt{P}_{=3}\,y$, this equational atom is *not* in a guard. Note that it makes no sense to restrict equational atoms only to guards since otherwise, they could be completely eliminated as has been described in subsection 4.3.2. Note also that symmetry of relations $x\,\mathtt{P}_{=1}\,y$,

$x \mathsf{P}_{=2}\, y$ and $x \mathsf{P}_{=3}\, y$ does not play an essential role in our reduction: we may assume that the paths are *directed* (for this purpose we have reversed a guard $y \mathsf{P}_{=1}\, x$ in conjuncts of GRID corresponded to diagonals).

Theorem 5.3.1. *$\mathcal{GF}^2_{\simeq}[\mathsf{CG}]$ is a conservative reduction class.*

5.3.2 Undecidability for Conjunctions of Transitive Guards

From subsection 5.2.2 we have seen that for the guarded fragment with equality, not only compositional atoms can be allowed as guards, but also their conjunctions. Since the extension of $\mathcal{GF}_{\simeq}$ with associative compositional guards is already undecidable, conjunctions of such atoms obviousely cannot be admiteted. However, there is still a hope that for conjunctions of *transitive* guards, i.e., for fragment $\mathcal{GF}_{\simeq}[\wedge\mathsf{TG}]$, decidability is retained. Unfortunately this is not true as we are going to demonstrate below.

For proving undecidability for $\mathcal{GF}_{\simeq}[\wedge\mathsf{TG}]$ we adapt a grid construction from [Ganzinger et al., 1999]. The idea is to split nodes on of a grid into equivalence classes modulo several equivalence relations such that (***i***) every cell of a grid fully belongs to some equivalence class and (***ii***) nodes that are "far enouph" belong to different equivalence classes. This makes it possible to "glue together" nodes that are close enouph in a grid, similarly as it has been done in the previous reductions.

What we exactly want to express is a grid structure illustrated in Figure 5.10. Here the "bricks" are the equivalence classeses induced by three transivie relations xSy, xTy and xHy. We create a "loop" using a transitive relation, and then "glue" the end nodes of this loop using equality (see the illustration in the left corner of Figure 5.10). Since every edge of a grid should belong to two equivalence classes, we should create these double edges at once. This can be done using *conjunctions of transitive guards* as shown in Figure 5.10. Again the essential part of this formula is the last three conjuncts which express the "gluing" operations. At this very moment we exploit transitivity of relations S, T and H.

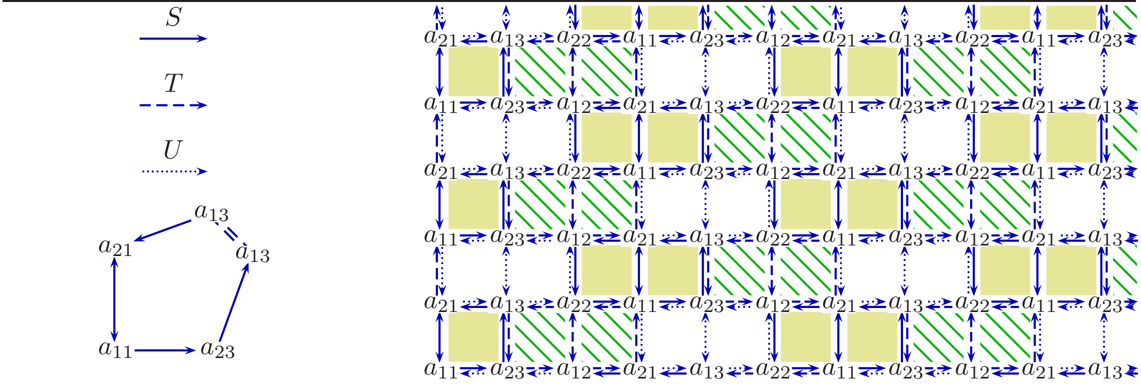
Theorem 5.3.2. *$\mathcal{GF}^2_{\simeq}[\wedge\mathsf{TG}]$ is a conservative reduction class.*

Note, that the same construction works when S, T and H are *equivalence* relations. This sharpens a recent result from [Kieronski & Otto, 2005] who demonstrated that $\mathcal{FO}^2$ with *three* built-in equivalence relations is undecidable:

Theorem 5.3.3. *The guarded fragment with conjunctions of equivalence guards $\mathcal{GF}^2_{\simeq}[\wedge\mathsf{EG}]$ is a conservative reduction class.*

Kieronski & Otto [2005] have also demonstrated that the $\mathcal{FO}^2$ with *two* equivalence relations remains *decidable*, so the above result is optimal w.r.t. the number of equivalence relations.

Figure 5.10 Undecidability of $\mathcal{GF}^2_{\simeq}$ with conjunctions of transitive guards


S
T
U
a_{13}
a_{21}
a_{13}
a_{11}
a_{23}
a_{21} a_{13} a_{22} a_{11} a_{23} a_{12} a_{21} a_{13} a_{22} a_{11} a_{23}
a_{11} a_{23} a_{12} a_{21} a_{13} a_{22} a_{11} a_{23} a_{12} a_{21} a_{13}
a_{21} a_{13} a_{22} a_{11} a_{23} a_{12} a_{21} a_{13} a_{22} a_{11} a_{23}
a_{11} a_{23} a_{12} a_{21} a_{13} a_{22} a_{11} a_{23} a_{12} a_{21} a_{13}
a_{21} a_{13} a_{22} a_{11} a_{23} a_{12} a_{21} a_{13} a_{22} a_{11} a_{23}
a_{11} a_{23} a_{12} a_{21} a_{13} a_{22} a_{11} a_{23} a_{12} a_{21} a_{13}

$\mathsf{GRID} \quad := \quad \exists x.a_{11}(x) \wedge$ - *creates the origin of a grid*

$\wedge \forall x.(a_{11}(x) \to \exists y.[xSy \wedge yUx \wedge H(x,y) \wedge a_{23}(y)]) \wedge$
$\wedge \forall x.(a_{23}(x) \to \exists y.[xTy \wedge yUx \wedge H(x,y) \wedge a_{12}(y)]) \wedge$
$\wedge \forall x.(a_{12}(x) \to \exists y.[xTy \wedge ySx \wedge H(x,y) \wedge a_{21}(y)]) \wedge$
$\wedge \forall x.(a_{21}(x) \to \exists y.[xUy \wedge ySx \wedge H(x,y) \wedge a_{13}(y)]) \wedge$
$\wedge \forall x.(a_{13}(x) \to \exists y.[xUy \wedge yTx \wedge H(x,y) \wedge a_{22}(y)]) \wedge$
$\wedge \forall x.(a_{22}(x) \to \exists y.[xSy \wedge yTx \wedge H(x,y) \wedge a_{11}(y)]) \wedge$
$\wedge \forall x.(a_{11}(x) \to \exists y.[xSy \wedge ySx \wedge V(x,y) \wedge a_{21}(y)]) \wedge$
$\wedge \forall x.(a_{21}(x) \to \exists y.[xTy \wedge yTx \wedge V(x,y) \wedge a_{22}(y)]) \wedge$
$\wedge \forall x.(a_{31}(x) \to \exists y.[xUy \wedge yUx \wedge V(x,y) \wedge a_{32}(y)]) \wedge$
$\wedge \forall x.(a_{21}(x) \to \exists y.[xTy \wedge yUx \wedge V(x,y) \wedge a_{11}(y)]) \wedge$
$\wedge \forall x.(a_{22}(x) \to \exists y.[xUy \wedge ySx \wedge V(x,y) \wedge a_{12}(y)]) \wedge$
$\wedge \forall x.(a_{23}(x) \to \exists y.[xSy \wedge yTx \wedge V(x,y) \wedge a_{13}(y)]) \wedge$

- *creates the double edges*

$\wedge \forall xy.[xSy \to ([a_{21}(x) \wedge a_{21}(y)] \vee [a_{13}(x) \wedge a_{13}(y)]) \to \boldsymbol{x} \simeq \boldsymbol{y}] \wedge$
$\wedge \forall xy.[xTy \to ([a_{22}(x) \wedge a_{22}(y)] \vee [a_{11}(x) \wedge a_{11}(y)]) \to \boldsymbol{x} \simeq \boldsymbol{y}] \wedge$
$\wedge \forall xy.[xUy \to ([a_{23}(x) \wedge a_{23}(y)] \vee [a_{12}(x) \wedge a_{12}(y)]) \to \boldsymbol{x} \simeq \boldsymbol{y}]$

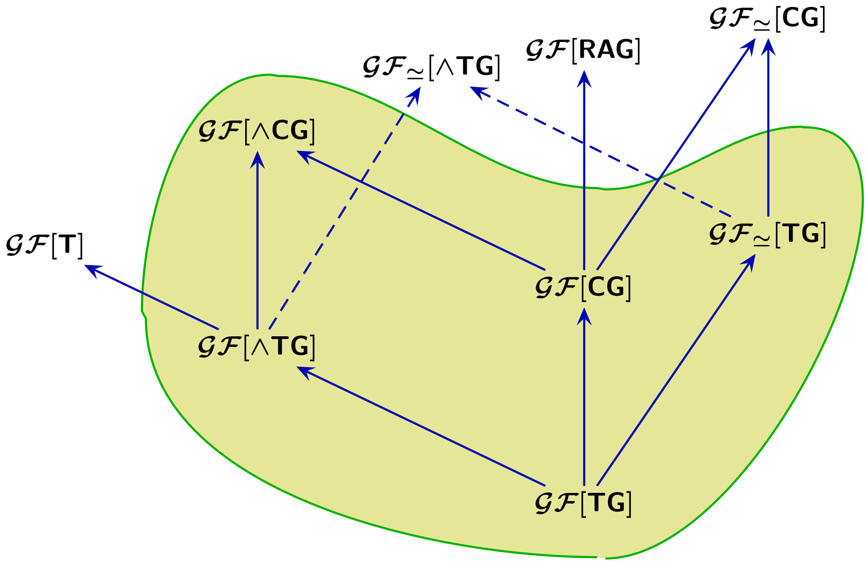
- *"glues" the cells*

5.3.3 A Decision Procedure for the Guarded Fragment with Transitive Guards and Equality

In the previous sections we have demonstrated that both the guarded fragment with compositional guards and the guarded fragment with conjunctions of transitive guards become undecidable with equality. However the full guarded fragment with transitive guards $\mathcal{GF}_{\simeq}[\mathsf{TG}]$ is decidable [Szwast & Tendera, 2001].

It is possible to extend our saturation-based decision procedure for this fragment, using additional simplification rules like Compositional Closure, to avoid some dangerous inferences with transitive relations and equality. However the resulted procedure is more complicated, since there is a large number of cases that should be

Figure 5.11 Summary of (un)decidabile extensions of $\mathcal{GF}$ with compositional theories



considered. Please see Appendix D.4 where we have highlighted the main principal ideas behind this decision procedure.

5.4 Conclusions and Future Works

In this chapter we have studied a variety of extensions of the guarded fragment with compositional axioms and have identified a border between "safe" and "dangerous" usage of compositional axioms. Our results are summarized in Figure 5.11. In this diagram the fragments that we have considered, are arranged in a lattice, where arrows represent inclusion between fragments. The extensions that are located within the enclosed area, are decidable. The others are not.

More precisely, the following results have been obtained in this chapter:

1. We *sharpened* the known *undecidability results* for the guarded fragment with transitivity [Grädel, 1999; Ganzinger et al., 1999] by demonstrating that the *two-variable guarded fragment* without equality is undecidable already with *two* transitive relations.

2. We gave several *chaining-based decision procedures* for extensions of the guarded fragment without equality, by compositional binary relations. The most

expressive of these fragments is the guarded fragment with *conjunctions of compositional guards*. All our procedures are optimal (2EXPTIME).

3. We proved that further extensions of the guarded fragment with compositional axioms are *undecidable*: (**1**) the *two-variable guarded fragment without equality*, with guards from a *relational algebra*; (**2**) the *two-variable guarded fragment with equality* and *compositional guards*; and (**3**) the *two-variable guarded fragment with equality* and *conjunctions of transitive guards*.

Below are some comments on these results:

Despite the fact that already some simple modal and description logics become *undecidable* with *general* compositional axioms (see subsection 2.5.1), this is not a reason to give up. We have found that decidability can be regained when restricting to a quite broad class of *associative* compositional axioms. Associativity is not an artificial condition, since most of the compositional theories that come from the "*real world*" (like various orderings, distance and path relations) are in fact *associative*. There is a close connection between associative compositional theories and *regular grammars*, which can be exploited in the future works.

We hoped that our methods can be extended to a larger class of compositional axioms of form $S \circ T \subseteq H_1 \cup \cdots \cup H_n$, which are related to spatial and interval-based temporal reasoning. It is well-known that such theories can be characterised as *relational algebras*, which admit many regular properties including *associativity*. Unfortunately, even with all these properties, extended compositional axioms might yield *undecidable* modal and description logics. But still it might be possible that extensions of modal and description logics with "interesting" relational algebras like *RCC-calculi* or *Allen's interval relations* are *decidable*. In this case one should probably identify some other regular properties of their compositional axioms, perhaps some variants of "acyclicity".

The guarded fragment with *conjunctions* of compositional guards $\mathcal{GF}[\wedge\mathsf{CG}]$ opens some other attractive perspectives for integration of useful theories into modal and description logics. For example, it is possible to express the *Allen's interval relations* using *conjunctions* of relations between endpoints of intervals of form $x <_{ll} y$, $x >_{rl} y$, $x =_{lr} y$, etc., which express that "*the left (right) endpoint of interval x is smaller (greater/equal) than the left (right) endpoint of interval y*". In contrast to relations between intervals, relations between endpoints admit *simple* compositional properties like $<_{ll} \circ <_{lr} \subseteq <_{lr}$, which are *associative*, plus *totality axioms*: $x <_{lr} y \vee x >_{lr} y \vee x =_{lr} y$, etc. Without totality axioms, the guarded fragment with "Allen's guards" could be mapped into $\mathcal{GF}[\wedge\mathsf{CG}]$ using this translation. A good question, is how to extend our decision procedures for *totality* axioms. Bachmair & Ganzinger [1998*b*, Section 6] have demonstrated that chaining inferences with totality axioms for *linear orders* are *redundant*, which might be a starting point for such an investigation.

Totality axioms are also the reason why our results cannot be yet applied to some simple associative theories like *theories of metric distances.* While a fully satisfactory solution for totality is still on its way, our results can be already used for some restricted versions of such theories, say to *distance relations* of form $x\,\mathsf{D}_{\leq n}\,y$, or to *path relations* of forms $x\ \mathsf{P}_{<n}\ y$, $x\ \mathsf{P}_{=n}\ y$ and $x\ \mathsf{P}_{>n}\ y$.

Finally *equality* again plays its vicious rôle by making most of the extensions with compositional axioms undecidable. However, this is not very surprising, since equality can be seen as a compositional relation which unlike other relations, makes no sense to restrict to guards only. In this light, the decidability result for the guarded fragment with transitive guards and equality $\mathcal{GF}_{\simeq}[\mathsf{TG}]$ turns out to be rather exceptional.

Chapter 6

Summary

In this thesis we have presented a variety of (un)decidability and complexity results, and decision procedures for reasoning in description logics and related fragments of first-order logic. We have considered a wide range of these formalisms: from a simple and tractable description logic $\mathcal{EL}$ to extensions of the guarded fragment with compositional axioms, which are 2EXPTIME-complete or even undecidable. Let us briefly highlight all these results and discuss possible directions for the future research.

DL $\mathcal{EL}$ and its extensions: It should not be regarded as a big success that a polynomial resolution-based decision procedure for $\mathcal{EL}$ has been found. The importance of this result is that it demonstrates how reasoning algorithms for description logics can be derived *formally* and *directly* from first-order translations of DL-constructors. Surprisingly, no theorem prover is required to implement this procedure, since it can be encoded as a datalog program. This on the one hand, gives a direct implementation, which outperforms general-purpose tableau reasoners. On the other hand, this allows one to apply a range of formal tools for optimisation and complexity analysis of such programs.

The basic procedure for $\mathcal{EL}$ has been extended to capture many other constructors: bottom concept, role hierarchies, conjunction of roles, cross-products of concepts, nominals and restricted role-value maps. For these constructors (not in all combinations) polynomial-time datalog programs have been derived. This confirms and generalises some recent results from [Baader et al., 2005].

$\mathcal{EL}$ turns out to be essentially the largest DL where one observes decidability with general role-value maps of form $S \circ T \sqsubseteq H$. It was shown that adding either conjunction of roles, disjunction of concepts, universal value restrictions or inverse roles, results in an undecidable logic.

Combinations of fragments: The idea of structural combination of decidable fragments by joining their constructors is rather natural if we try to mimic description logics. It became a relatively simple consequence of our uniform presentation of resolution decision procedures for the guarded, two-variable and monadic fragments, that the structural combinations of these fragment are decidable by resolution. Combinations of these fragment are not only decidable – they retain the maximal complexity of their components. Unfortunately these results do not extend to equality, but still the new notion of combination might be useful for identifying more expressive decidable first-order fragments.

Extensions of $\mathcal{GF}$ with constants and counting: Two paramodulation-based decision procedures have been described for the guarded fragment with constants. First procedure uses elimination of equational guards followed by Grädel's [1999] elimination of constants, and is relatively straightforward. The second procedure does not employ any transformation, and is more involved since it uses splitting with eager ground rewriting. Both procedures give essentially the same complexity showing that the (bounded variable) guarded fragment with constants is decidable in 2EXPTIME (EXPTIME).

Recall that the guarded fragments with functionality and number restrictions have been considered in connection with description logics with functional roles, respectively, with (qualified) number restrictions. Because of a general undecidability result for the guarded fragment with functionality $\mathcal{GF}[\mathsf{F}]$ [Grädel, 1999], we have considered a restriction of $\mathcal{GF}[\mathsf{F}]$ where functional atoms, respectively counting atoms, may appear in guards only: $\mathcal{GF}[\mathsf{FG}]$ and $\mathcal{GFN}$. For these fragments, optimal paramodulation-based decision procedures have been formulated (in the case with number restrictions, assuming the unary coding of numbers).

It is worth noting that the guarded fragment with counting guards does not capture description logics with role hierarchies, like $\mathcal{ALCQIH}$, since not all roles in role inclusion axioms are used as guards. For such description logics, the fragment $\mathcal{GF}^2\mathcal{N}$ considered in [Kazakov, 2004] can be used.

The paramodulation-based decision procedure for $\mathcal{GFN}$ does not give us satisfactory complexity when binary coding of numbers is assumed. This is because the first-order translation of number restrictions is exponential when numbers are coded in binary. Perhaps some ideas from [Kazakov, 2004] can be adapted to find optimal translation and decision procedure.

Extensions of $\mathcal{GF}$ with compositional axioms: The most technically involved result of this thesis is a classification of extensions of the guarded fragment with compositional axioms of form $S \circ T \subseteq H$. As has been mentioned above, some simple description logics become already undecidable with general axioms of this form. However, in many cases such axioms can be integrated into logical formalisms:

Despite general undecidability results for the guarded fragment with transitivity $\mathcal{GF}[\mathsf{T}]$ [Grädel, 1999; Ganzinger et al., 1999], which we have sharpened to $\mathcal{GF}^2[\mathsf{T}]$ with two transitive relations, a restricted fragment $\mathcal{GF}[\mathsf{TG}]$ where transitive relations appear only in guards, is decidable [Szwast & Tendera, 2001]. We have found a chaining-based procedure for $\mathcal{GF}[\mathsf{TG}]$ without equality, which uses a special simplification rule Transitive Closure. This is a novel simplification rule based on an advanced notion of redundancy – redundancy of inferences.

A careful inspection of our procedure has revealed that it can be actually extended to a larger class of guarded formulas: transitivity can be generalised to arbitrary *associative* compositional axioms, and conditions on occurrences of compositional atoms can be relaxed by admitting conjunctions of such atoms in guard positions. Such compound guards can be used to characterise Allen's interval relations, which can be represented as conjunctions of relations admitting simple associative compositional axioms.

The above extensions were essentially the border of decidability: we have demonstrated that generalised compositional axioms of form $S \circ T \subseteq H_1 \cup \cdots \cup H_n$, which admit many "natural" algebraic properties, cannot be integrated even into the modal fragment, and extensions of the guarded fragment with compositional guards and conjunctions of transitive guards, do not remain decidable with equality. The only fragment that remains decidable with equality is the guarded fragment with transitive guards $\mathcal{GF}_{\simeq}[\mathsf{TG}]$ [Szwast & Tendera, 2001], for which we have sketched a chaining-based strategy using additional simplification rules.

On perspectives

This thesis presents many new decision procedures and undecidability results. However we think that the main value of this thesis is the techniques which allow one to design saturation-based decision procedures for a broad range of modal, description logics and first-order fragments.

In particular, by combining our decision procedures for extensions of the guarded fragment with constants, number restrictions and transitivity, it might be possible to come up with a reasoning procedure for DL $\mathcal{SHOIN}$ which corresponds to the ontology language OWL for the semantic web [Horrocks & Patel-Schneider, 2004].

Before starting to implement our decision procedures one should first address certain issues revealed in our first experiments with DL $\mathcal{EL}$. Saturation based procedures by their very nature, are opposed to search-based procedures, like tableau-based ones. They win in time, but they lose in memory. This is one of the reasons why there are so far no optimal saturation-based procedures for PSPACE-complete reasoning problems, like satisfiability of $\mathcal{ALC}$-concepts: the time complexity of deterministic saturation procedures is the same as their space complexity. Hence a challenging direction for the future work could be in integration of search strategies into saturation-based theorem proving. In particular, a combination of resolution with instance-based methods [Letz & Stenz, 2001; Ganzinger & Korovin, 2003; Baumgartner & Tinelli, 2003] might bear some fruits. Alternatively, the space consumption of particular saturation-based procedures might be reduced by using indexing techniques like BDDs [like in Pan, Sattler & Vardi, 2002]. Memory optimisation of completion-based procedures for $\mathcal{EL}$ discussed in chapter 2 could open new possibilities for practical reasoning with very large terminologies like SNOMED.

Our procedures for extensions of the guarded fragment with compositional theories, can be specialised to particular description logics with complex role dependencies. As has been pointed in the end of subsection 5.2.1, many description logics do not require Transitive Closure or Compositional Closure rules in their full generality. Many of such logics might be translated to $\mathcal{ALCI}$ by employing an extended version of de Nivelle's [1999] translation (say the logics of metric distances studied in [Wolter & Zakharyaschev, 2003]). A particular challenge is to extend our techniques to compositional relations that additionally admit totality axioms, like distance relations $x\,\mathtt{D}_{\leq n}\,y$, $x\,\mathtt{D}_{<n}\,y$, $x\,\mathtt{D}_{\geq n}\,y$ and $x\,\mathtt{D}_{>n}\,y$ and Allen's interval relations.

Some proofs in our thesis pose few new questions. For example, the undecidability proof for $\mathcal{GF}[\mathsf{T}]$ given in subsection 5.1.3 does not work for one transitive relation, which suggests that the guarded fragment over *one* transitive relation might be decidable. In some applications, one transitive relation suffices, for example for temporal reasoning. The guarded fragment with one transitive relation can express new properties, say the *density axiom*: $\forall xy.(x < y \rightarrow \exists z.[\mathsf{Between}(z, x, y)]) \wedge \forall x, y, z.[\mathsf{Between}(z, x, y) \rightarrow x < z \wedge z < y]$. The same question is about the guarded fragment with *two equivalence* relations, related to the undecidability result from subsection 5.3.2 (the two-variable fragment with two equivalence relations was recently shown to be decidable in [Kieronski & Otto, 2005]).

Finally, certain techniques demonstrated in this thesis might be useful for automated reasoning in general. In particular the Literal Projection rule as a controlled version of Splitting through New Predicate Symbol, can be applied in many cases to keep the term depth from growing (see subsection 4.3.3). These were essentially refinements of general saturation procedures, which made it possible to obtain many decidability results. We think that in return, the lessons learned from our decision procedures, may influence the automated reasoning in first-order logic, since the quality and success of a general theorem prover depends highly on its ability to decide expressive first-order fragments.

Appendix A

DL $\mathcal{EL}$ and Its Extensions

A.1 Evaluation of Queries in DL $\mathcal{EL}$ Using Ordered Resolution

In this Appendix we demonstrate how subsumption of concepts can be derived for our simple terminology defined in Table 2.1 using the ordered resolution calculus. Recall that in section 2.1 we have proved by our “logical reasoning” that concept Parent subsumes concept Father, which subsumes concept Grandfather. Now we verify our reasoning formally, using resolution theorem proving.

In order to demonstrate the first subsumption relation, we first translate the definitions for concepts Man, Parent and Father from (2.3) according to Table 2.3:

C1 $\neg\underline{\mathsf{Man}(x)} \vee \mathsf{Human}(x)$;
C2 $\neg\underline{\mathsf{Man}(x)} \vee \mathsf{Male}(x)$;
C3 $\neg\underline{\mathsf{Human}(x)} \vee \neg\mathsf{Male}(x) \vee \mathsf{Man}(x)$;
C4 $\neg\underline{\mathsf{Father}(x)} \vee \mathsf{Man}(x)$;
C5 $\neg\underline{\mathsf{Father}(x)} \vee \mathsf{A}(x)$;
C6 $\neg\underline{\mathsf{Man}(x)} \vee \neg\mathsf{A}(x) \vee \mathsf{Father}(x)$;
C7 $\neg\underline{\mathsf{Parent}(x)} \vee \mathsf{Human}(x)$;
C8 $\neg\underline{\mathsf{Parent}(x)} \vee \mathsf{A}(x)$;
C9 $\neg\underline{\mathsf{Human}(x)} \vee \neg\mathsf{A}(x) \vee \mathsf{Parent}(x)$;

Concept subsumption query ?- Father $\sqsubseteq$ Parent is translated according to Table 2.4 to clauses:

Q1 $\underline{\mathsf{Father}(\mathsf{c})}$;
Q2 $\neg\underline{\mathsf{Parent}(\mathsf{c})}$;

where c is a fresh (Skolem) constant. For the above clause set we apply a resolution strategy, according to which, we select the first negative literal in every clause, if there is one:

$\mathtt{OR}[\mathsf{Q1};\mathsf{C4}]: \underline{\mathsf{Father}(\mathtt{c})};\ \neg\underline{\mathsf{Father}(x)} \vee \mathsf{Man}(x)\ \vdash \quad \mathsf{Q3}\ \underline{\mathsf{Man}(\mathtt{c})};$

$\mathtt{OR}[\mathsf{Q1};\mathsf{C5}]: \underline{\mathsf{Father}(\mathtt{c})};\ \neg\underline{\mathsf{Father}(x)} \vee \mathsf{A}(x)\ \vdash \quad \mathsf{Q4}\ \underline{\mathsf{A}(\mathtt{c})};$

$\mathtt{OR}[\mathsf{Q3};\mathsf{C1}]: \underline{\mathsf{Man}(\mathtt{c})};\ \neg\underline{\mathsf{Man}(x)} \vee \mathsf{Human}(x)\ \vdash \quad \mathsf{Q5}\ \underline{\mathsf{Human}(\mathtt{c})};$

$\mathtt{OR}[\mathsf{Q5};\mathsf{C9}]: \underline{\mathsf{Human}(\mathtt{c})};\ \neg\underline{\mathsf{Human}(x)} \vee \neg\mathsf{A}(x) \vee \mathsf{Parent}(x)\ \vdash \quad \mathsf{Q6}\ \neg\underline{\mathsf{A}(\mathtt{c})} \vee \mathsf{Parent}(\mathtt{c});$

$\mathtt{OR}[\mathsf{Q4};\mathsf{Q6}]: \underline{\mathsf{A}(\mathtt{c})};\ \neg\underline{\mathsf{A}(\mathtt{c})} \vee \mathsf{Parent}(\mathtt{c})\ \vdash \quad \mathsf{Q7}\ \underline{\mathsf{Parent}(\mathtt{c})};$

$\mathtt{OR}[\mathsf{Q7};\mathsf{Q2}]: \underline{\mathsf{Parent}(\mathtt{c})};\ \neg\underline{\mathsf{Parent}(\mathtt{c})}\ \vdash \quad \bot\ \ \square.$

We have obtained a contradiction, which means that subsumption between concepts Father and Parent is derivable from TBox.

In order to prove the second subsumption relation, we add the **CNF**-translations for the remaining concepts A, B and Grandfather:

$\mathsf{C10}\ \neg\mathsf{A}(x) \vee \underline{\mathsf{has\text{-}child}(x,\mathtt{f}(x))};$

$\mathsf{C11}\ \neg\mathsf{A}(x) \vee \underline{\mathsf{Human}(\mathtt{f}(x))};$

$\mathsf{C12}\ \neg\underline{\mathsf{has\text{-}child}(x,y)} \vee \neg\mathsf{Human}(y) \vee \mathsf{A}(x);$

$\mathsf{C13}\ \neg\mathsf{B}(x) \vee \underline{\mathsf{has\text{-}child}(x,\mathsf{g}(x))};$

$\mathsf{C14}\ \neg\mathsf{B}(x) \vee \underline{\mathsf{Parent}(\mathsf{g}(x))};$

$\mathsf{C15}\ \neg\underline{\mathsf{has\text{-}child}(x,y)} \vee \neg\mathsf{Parent}(y) \vee \mathsf{B}(x);$

$\mathsf{C16}\ \neg\underline{\mathsf{Grandfather}(x)} \vee \mathsf{Man}(x);$

$\mathsf{C17}\ \neg\underline{\mathsf{Grandfather}(x)} \vee \mathsf{B}(x);$

$\mathsf{C18}\ \neg\underline{\mathsf{Man}(x)} \vee \neg\mathsf{B}(x) \vee \mathsf{Grandfather}(x);$

where `f` and `g` are fresh Skolem functions. The query ?- Grandfather $\sqsubseteq$ Father is translated to clauses:

$\mathsf{S1}\ \ \underline{\mathsf{Grandfather}(\mathtt{d})};$

$\mathsf{S2}\ \neg\underline{\mathsf{Father}(\mathtt{d})};$

where `d` is a fresh Skolem constant introduced for this query. We need to derive the empty clause from clauses C1 – C18 and S1, S2. For this, we modify our resolution strategy. Now we select the first negative literal only in clauses that do not contain functional symbols. In clauses with functional symbols, namely C10, C11, C13, C14 and perhaps other clauses that will be derived, we select no literals, so we have to resolve on its maximal literals (which are underlined). Proceeding according this strategy, we obtain the following inferences:

$\mathtt{OR}[\mathsf{Q1};\mathsf{C5}]\ : \underline{\mathsf{Father}(\mathtt{c})};\ \neg\underline{\mathsf{Father}(x)} \vee \mathsf{A}(x)\ \vdash \quad \mathsf{Q4}\ \underline{\mathsf{A}(\mathtt{c})};$

$\mathtt{OR}[\mathsf{C14};\mathsf{C7}]\ : \neg\mathsf{B}(x) \vee \underline{\mathsf{Parent}(\mathsf{g}(x))};\ \neg\underline{\mathsf{Parent}(x)} \vee \mathsf{Human}(x)\ \vdash$

$\vdash\ \mathsf{C19}\ \neg\mathsf{B}(x) \vee \underline{\mathsf{Human}(\mathsf{g}(x))};$

$\mathtt{OR}[\mathsf{C13};\mathsf{C12}]: \neg\mathsf{B}(x) \vee \underline{\mathsf{has\text{-}child}(x,\mathsf{g}(x))};\ \neg\underline{\mathsf{has\text{-}child}(x,y)} \vee \neg\mathsf{Human}(y) \vee \mathsf{A}(x)$

$\vdash\ \mathsf{C20}\ \neg\mathsf{B}(x) \vee \neg\underline{\mathsf{Human}(\mathsf{g}(x))} \vee \mathsf{A}(x);$

$\mathtt{OR}[\mathsf{C19};\mathsf{C20}]: \neg\mathsf{B}(x) \vee \underline{\mathsf{Human}(\mathsf{g}(x))};\ \neg\mathsf{B}(x) \vee \neg\underline{\mathsf{Human}(\mathsf{g}(x))} \vee \mathsf{A}(x)\ \vdash$

Continued on next page

$$\begin{array}{lll}
 & \vdash \neg\mathsf{B}(x) \vee \neg\mathsf{B}(x) \vee \mathsf{A}(x) \ \vdash & \mathsf{C21}\ \neg\underline{\mathsf{B}(x)} \vee \mathsf{A}(x); \\
\mathtt{OR}[\mathsf{S1};\mathsf{C16}] & : \underline{\mathsf{Grandfather}(\mathtt{d})};\ \neg\underline{\mathsf{Grandfather}(x)} \vee \mathsf{Man}(x) \ \vdash & \mathsf{S3}\ \ \underline{\mathsf{Man}(\mathtt{d})}; \\
\mathtt{OR}[\mathsf{S1};\mathsf{C17}] & : \underline{\mathsf{Grandfather}(\mathtt{d})};\ \neg\underline{\mathsf{Grandfather}(x)} \vee \mathsf{B}(x) \ \vdash & \mathsf{S4}\ \ \underline{\mathsf{B}(\mathtt{d})}; \\
\mathtt{OR}[\mathsf{S4};\mathsf{C21}] & : \underline{\mathsf{B}(\mathtt{d})};\ \neg\underline{\mathsf{B}(x)} \vee \mathsf{A}(x) \ \vdash & \mathsf{S5}\ \ \underline{\mathsf{A}(\mathtt{d})}; \\
\mathtt{OR}[\mathsf{S3};\mathsf{C6}] & : \underline{\mathsf{Man}(\mathtt{d})};\ \neg\underline{\mathsf{Man}(x)} \vee \neg\mathsf{A}(x) \vee \mathsf{Father}(x) \ \vdash & \mathsf{S6}\ \ \neg\underline{\mathsf{A}(\mathtt{d})} \vee \mathsf{Father}(\mathtt{d}); \\
\mathtt{OR}[\mathsf{S5};\mathsf{S6}] & : \underline{\mathsf{A}(\mathtt{d})};\ \neg\underline{\mathsf{A}(\mathtt{d})} \vee \mathsf{Father}(\mathtt{d}) \ \vdash & \mathsf{S7}\ \ \underline{\mathsf{Father}(\mathtt{d})}; \\
\mathtt{OR}[\mathsf{S7};\mathsf{S2}] & : \underline{\mathsf{Father}(\mathtt{d})};\ \neg\underline{\mathsf{Father}(\mathtt{d})} \ \vdash & \bot\ \ \Box.
\end{array}$$

Note that we have used the Elimination of Duplicate Literals rule in inference $\mathtt{OR}[\mathsf{C19};\mathsf{C20}]$ to eliminate duplicate occurrences of literal $\neg\mathsf{B}(x)$ in its conclusion.

Finally, we demonstrate that individual *John* is a Grandfather under ABox-assertions from Table 2.1. Applying translation for ABox according to Table 2.4, we obtain clauses:

$\mathsf{A1}\ \underline{\mathsf{Man}(John)};$
$\mathsf{A2}\ \underline{\mathsf{Father}(Bill)};$
$\mathsf{A3}\ \underline{\mathsf{has\text{-}child}(John, Bill)};$

Query ?- *John* : Grandfather is translated according to Table 2.4 to clause:

$\mathsf{I1}\ \neg\underline{\mathsf{Grandfather}(John)};$

Repeating the same inferences as for the first query ?- Father $\sqsubseteq$ Parent with c replaced with *Bill*, and using A2 instead of Q1 we obtain clause:

$\mathsf{I2}\ \underline{\mathsf{Parent}(Bill)};$

Now, the above clauses can be refuted as follows:

$$\begin{array}{lll}
\mathtt{OR}[\mathsf{A3};\mathsf{C15}] & : \underline{\mathsf{has\text{-}child}(John, Bill)};\ \neg\underline{\mathsf{has\text{-}child}(x,y)} \vee \neg\mathsf{Parent}(y) \vee \mathsf{B}(x) \ \vdash & \\
 & \qquad \vdash\ \mathsf{A4}\ \neg\underline{\mathsf{Parent}(Bill)} \vee \mathsf{B}(John); & \\
\mathtt{OR}[\mathsf{I2};\mathsf{A4}] & : \underline{\mathsf{Parent}(Bill)};\ \neg\underline{\mathsf{Parent}(Bill)} \vee \mathsf{B}(John) \ \vdash & \mathsf{I3}\ \underline{\mathsf{B}(John)}; \\
\mathtt{OR}[\mathsf{A1};\mathsf{C18}] & : \underline{\mathsf{Man}(John)};\ \neg\underline{\mathsf{Man}(x)} \vee \neg\mathsf{B}(x) \vee \mathsf{Grandfather}(x) \ \vdash & \\
 & \qquad \vdash\ \mathsf{A5}\ \neg\underline{\mathsf{B}(John)} \vee \mathsf{Grandfather}(John); & \\
\mathtt{OR}[\mathsf{I3};\mathsf{A5}] & : \underline{\mathsf{B}(John)};\ \neg\underline{\mathsf{B}(John)} \vee \mathsf{Grandfather}(John) \ \vdash & \mathsf{I4}\ \underline{\mathsf{Grandfather}(John)}; \\
\mathtt{OR}[\mathsf{I4};\mathsf{I1}] & : \underline{\mathsf{Grandfather}(John)};\ \neg\underline{\mathsf{Grandfather}(John)} \ \vdash & \bot\ \Box.
\end{array}$$

Well, the above proofs are slightly more complicated than those we sketched using our "logical reasoning". However, they are purely formal and can be implemented using a computer.

A.2 An Example for Query Evaluation in DL $\mathcal{EL}$ Using Datalog

In this appendix we continue an example from subsection 2.3.2, demonstrating how to query our sample terminology of human relations (2.3) using the reduction to datalog.

Recall that in Table 2.8 we have computed translation for definitions, assertions and queries for our example, according to Table 2.3 and Table 2.4. Now we need to compute a deductive closure for the resulted sets of atoms under program in Table 2.7 to answer queries. First we compute a deductive closure for the set of atoms that correspond to TBox and ABox, and then we show how $\perp$ is obtained after adding atoms that correspond to each query.

In Table A.3 we have computed a deductive closure for TBox and ABox. Each table C1 – C5 and D3 – D8 represents the values of respective predicates, which where computed as follows.

The values indicated by (0) are taken from the translation in Table 2.8. After that, we iteratively apply inferences A1 – A8 and T1 – T9 from Table 2.3 to derive new values of predicates from old ones. For example, values (1) of predicate C4 are computed using rule T5 from values (0) of C4 and values (0) of D4 (see Table A.3). To compute new values according to this rule, we have to match the values in the first column of both tables and take the remaining two values as an answer. This process is repeated until nothing new can be derived. At this moment a deductive closure of the database for TBox and ABox is computed.

Table A.3: A deductive closure for the database correspondent to TBox and ABox

C4	A	B	f_A
(0)	A	Human	f
	B	Parent	g
(1)	T5[C4(0), D4(0)]:		
	B	Human	g
	B	A	g

C5	A	R	f_A
(0)	A	has-child	f
	B	has-child	g

C2	R	a	b
(0)	has-father	*John*	Bill

D4	A	B
(0)	Man	Human
	Man	Male
	Parent	Human
	Parent	A
	Father	Man
	Father	A
	Grandfather	Man
	Grandfather	B
(1)	T7[C4(0,1), D7(0)]:	
	A	A
	B	A
	B	B

D5	A	B	C
(0)	Human	Male	Man
	Human	A	Parent
	Man	A	Father
	Man	B	Grandfather

D6	R	A	B
(0)	has-child	Human	A
	has-child	Parent	B

Continued on next page

C1	*A*	*a*
(0)	Man	*John*
	Father	*Bill*
(1)	A3[C1(0), D4(0, 1)] :	
	Human	*John*
	Male	*John*
	Man	*Bill*
	A	*Bill*
(2)	A3[C1(1), D4(0, 1)] :	
	Human	*Bill*
	Male	*Bill*
(3)	A2[C1(0, 1, 2), D3(1, 2)] :	
	Parent	*Bill*
	A	*John*
(4)	A2[C1(3), D3(1, 2)] :	
	Parent	*John*
	Father	*John*
	B	*John*
(5)	A2[C1(4), D3(1, 2)] :	
	Grandfather	*John*

D7	*A*	*B*	*C*	f_A
(1)	T9[C5(0), D6(0)]:			
	A	Human	A	f
	A	Parent	B	f
	B	Human	A	g
	B	Parent	B	g

D8	*A*	*B*	*C*	f_A
(1)	T6[C4(0,1), D5(0)]:			
	A	Male	Man	f
	A	A	Parent	f
	B	Male	Man	g
	B	A	Parent	g

D3	*A*	*B*	*a*	*b*
(1)	A4[C1(0, 1), D5(0)] :			
	Male	Man	*John*	*John*
	Male	Man	*Bill*	*Bill*
	A	Parent	*John*	*John*
	A	Parent	*Bill*	*Bill*
	A	Father	*John*	*John*
	A	Father	*Bill*	*Bill*
	B	Grandfather	*John*	*John*
	B	Grandfather	*Bill*	*Bill*
(2)	A6[C2(0), D6(0)] :			
	Human	A	*Bill*	*John*
	Parent	B	*Bill*	*John*

To answer the query ?- Grandfather $\sqsubseteq$ Father we add two correspondent ground atoms according Table 2.8, and proceed further computing new inferences with these atoms and computed database for ABox and TBox: see Table A.4.

Table A.4: Answering of the queries ?- Grandfather $\sqsubseteq$ Father and ?- *John* : Grandfather

C1	*A*	*a*
(S.0)	Grandfather	c
(S.1)	A3[C1(S.0), D4(0, 1)] :	
	Man	c
	B	c
(S.2)	A3[C1(S.1), D4(0, 1)] :	
	Human	c
	Male	c
	A	c
(S.3)	C1[C1(S.0, S.1, S.2), D3(S.1)] :	
	Father	c
	Parent	c

D3	*A*	*B*	*a*	*b*
(S.1)	A4[C1(S.0, S.1, S.2), D5(0)] :			
	A	Father	c	c
	B	Grandfather	c	c
	Male	Man	c	c
	A	Parent	c	c

D1	*A*	*a*
(S.0)	Father	c
(I.0)	Grandfather	*John*

$\bot$	
(S.1)	A1[C1(S.3), D1(S.0)]
(I.1)	A1[C1(5), D1(I.0)]

We indicate all new instances that where produced for our subsumption query with prefix "S". As seen from Table A.4, there is an S-inference producing predicate $\bot$. So our query is answered *positively*.

To answer the other query $\texttt{?-}\ John : \mathsf{Grandfather}$, we discard all inferences done for the previous query, and compute deductive closure of TBox and ABox with additional atom $\mathsf{D1}(I.0)$ that corresponds to this **i**nstance problem. Processing of this particular query is especially easy, since the fact $\mathsf{C1}(5)$ computed for ABox and TBox immediately yields $\bot$. Hence our instance query is answered *positively* as well.

A.3 Additional Datalog Rules for Querying Subsumption Relation in DL $\mathcal{EL}$

In this appendix, we return to subsection 2.3.2 and demonstrate how to *derive* additional datalog rules stated in Table 2.10, using which the explicit subsumption relation in $\mathcal{EL}$ can be computed.

Recall that for checking subsumption $\texttt{?-}\ A \sqsubseteq B$, we have to derive $\bot$ using atoms $\mathsf{C1}(A, c)$ and $\mathsf{D1}(B, c)$, where $c = c(A, B)$ is a *fresh* constant for this query (see Table 2.4). For the similar reasons as for the instance query, $\bot$ can be derived only using rules $\mathsf{A1}$ or $\mathsf{A7}$. Hence for showing subsumption $A \sqsubseteq B$, we have to derive either $\mathsf{C1}(B, c(A, B))$ or $\mathsf{C3}(B)$ using additional assumption $\mathsf{C1}(A, c(A, B))$. Hence, the subsumption relation can be explicitly computed by the following extension of our program from Table 2.7:

$$\begin{array}{ll} \mathsf{S0.1} & \mathsf{C1}(A, c(A,B)) \ \leftarrow . \\ \mathsf{S1.1} & \mathsf{subsumes}(B, A) \leftarrow \mathsf{C3}(B). \\ \mathsf{S2.1} & \mathsf{subsumes}(B, A) \leftarrow \mathsf{C1}(B, c(A,B)). \end{array} \tag{A.1}$$

Although the resulted program can, in principle, be used to query subsumption, it is not a *datalog* program because of $c(A, B)$, which is now considered as a *function* that assigns a constant to concept names A and B (i.e., to concept subsumption problem). Hence we would like to transform program (A.1) into an equivalent *datalog* program.

In order to transform the program (A.1), we consider instances of inferences from Table 2.7, using which atoms with functional terms can be produced. Atom $\mathsf{C1}(B, c(A, B))$ can be derived by rules $\mathsf{A2}$, $\mathsf{A3}$ or $\mathsf{A8}$. By specialising the heads of these rules for our atom, we obtain rules:

$$\begin{array}{ll} \mathsf{A2.1} & \mathsf{C1}(B, c(A,B)) \leftarrow \mathsf{C1}(C, a),\ \mathsf{D3}(C, B, a, c(A,B)). \\ \mathsf{A3.1} & \mathsf{C1}(B, c(A,B)) \leftarrow \mathsf{C1}(C, c(A,B)),\ \mathsf{D4}(C, B). \\ \mathsf{A8.1} & \mathsf{C1}(B, c(A,B)) \leftarrow \mathsf{C3}(C),\ \mathsf{D3}(C, B, a, c(A,B)). \end{array} \tag{A.2}$$

The subgoal $\mathsf{D3}(D, C, a, c(A, B))$ of rules A2.1 and A8.1 can be obtained only by rules A4 and A6. However, A6 cannot produce $\mathsf{D3}(C, B, a, c(A, B))$ since there is no atom of form $\mathsf{C2}(R, c(A, B), a)$ in the initial database and those cannot be produced (recall that $c(A, B)$ is a *fresh* constant, so it may not be present in role assertions). By unfolding $\mathsf{D3}(C, B, a, c(A, B))$ using rule A4, we simplify (A.2) as follows:

$$\begin{array}{ll}
\mathsf{A2.2} & \mathsf{C1}(B, c(A, B)) \leftarrow \mathsf{C1}(C, c(A, B)),\ \mathsf{C1}(D, c(A, B)),\ \mathsf{D5}(D, C, B). \\
\mathsf{A3.2} & \mathsf{C1}(B, c(A, B)) \leftarrow \mathsf{C1}(C, c(A, B)),\ \mathsf{D4}(C, B). \\
\mathsf{A8.2} & \mathsf{C1}(B, c(A, B)) \leftarrow \mathsf{C3}(C),\ \mathsf{C1}(D, c(A, B)),\ \mathsf{D5}(D, C, B).
\end{array} \tag{A.3}$$

Now we proceed to subgoals $\mathsf{C1}(C, c(A, B))$ which can also be obtained only by rules A2, A3 and A8. Repeating similar steps as above we obtain rules that are more general than (A.3):

$$\begin{array}{ll}
\mathsf{A2.3} & \mathsf{C1}(E, c(A, B)) \leftarrow \mathsf{C1}(C, c(A, B)),\ \mathsf{C1}(D, c(A, B)),\ \mathsf{D5}(D, C, E). \\
\mathsf{A3.3} & \mathsf{C1}(E, c(A, B)) \leftarrow \mathsf{C1}(C, c(A, B)),\ \mathsf{D4}(C, E). \\
\mathsf{A8.3} & \mathsf{C1}(E, c(A, B)) \leftarrow \mathsf{C3}(C),\ \mathsf{C1}(D, c(A, B)),\ \mathsf{D5}(D, C, E).
\end{array} \tag{A.4}$$

Now every instance of rules from Table 2.7 that can be used to produce an atom of form $\mathsf{C1}(C, c(A, B))$ is among (A.4) and we have separated a non-datalog part of our program from a datalog part. The trick now is to treat $\mathsf{C1}(A, c(B, C))$ as a new relation $\mathsf{sb}(A, B, C)$. In this way, the subsumption relation can be computed by performing this replacement in (A.2) and (A.4):

$$\begin{array}{lll}
\mathsf{S0.2} & \mathsf{sb}(A, A, B) & \leftarrow . \\
\mathsf{S1.2} & \mathsf{subsumes}(B, A) & \leftarrow \mathsf{C3}(B). \\
\mathsf{S2.2} & \mathsf{subsumes}(B, A) & \leftarrow \mathsf{sb}(B, A, B). \\
\mathsf{S3.2} & \mathsf{sb}(E, A, B) & \leftarrow \mathsf{sb}(C, A, B),\ \mathsf{D4}(C, E). \\
\mathsf{S4.2} & \mathsf{sb}(E, A, B) & \leftarrow \mathsf{sb}(C, A, B),\ \mathsf{sb}(D, A, B),\ \mathsf{D5}(D, C, E). \\
\mathsf{S5.2} & \mathsf{sb}(E, A, B) & \leftarrow \mathsf{C3}(C),\ \mathsf{sb}(D, A, B),\ \mathsf{D5}(D, C, E).
\end{array} \tag{A.5}$$

However this is not the end of our story, as we may notice that in any bottom-up derivation the last argument of predicate $\mathsf{sb}(*, *, *)$ is never instantiated. Hence it might be completely dropped and the following program computes the same subsumption relation as (A.5):

$$\begin{array}{lll}
\mathsf{S0} & \mathsf{sb}(A, A) & \leftarrow . \\
\mathsf{S1} & \mathsf{subsumes}(B, A) & \leftarrow \mathsf{C3}(B). \\
\mathsf{S2} & \mathsf{subsumes}(B, A) & \leftarrow \mathsf{sb}(B, A). \\
\mathsf{S3} & \mathsf{sb}(E, A) & \leftarrow \mathsf{sb}(C, A),\ \mathsf{D4}(C, E). \\
\mathsf{S4} & \mathsf{sb}(E, A) & \leftarrow \mathsf{sb}(C, A),\ \mathsf{sb}(D, A),\ \mathsf{D5}(D, C, E). \\
\mathsf{S5} & \mathsf{sb}(E, A) & \leftarrow \mathsf{C3}(C),\ \mathsf{sb}(D, A),\ \mathsf{D5}(D, C, E).
\end{array} \tag{A.6}$$

A.4 Extensions of DL $\mathcal{EL}$ with Cross-Products of Concepts

In this appendix we return to subsection 2.4.2 and describe a resolution decision procedure for extension of DL $\mathcal{EL}$ with *cross-products of concepts*. We demonstrate that resolution inferences produce either clauses for $\mathcal{EL}$ or clauses of new types listed in Table A.5. Indeed, in Table A.6 we have enumerated all possible inferences between clauses for $\mathcal{EL}$ from Table 2.5 and new clauses for cross-products of concepts from Table A.5. In the end of this table we have also listed all inferences with clause DH1 from Table 2.13 which corresponds to an extension of $\mathcal{EL}$ with simple role inclusion axioms. Table A.6 proves that the mentioned types of clauses are closed under inferences of ordered resolution, which implies that reasoning in the extension of $\mathcal{EL}$ with simple role hierarchies and cross-products of concepts can be done in polynomial time. This table naturally yields a datalog program given in Table A.7 for reasoning in this description logic.

Table A.5 Additional clause types for the extension of DL $\mathcal{EL}$ with cross-**p**roducts of concepts

CP1	$\underline{\boldsymbol{R}(x,\boldsymbol{a})}$;	DP1	$\neg\underline{\boldsymbol{A}(\boldsymbol{a})} \vee \boldsymbol{B}(x)$;
CP2	$\underline{\boldsymbol{R}(\boldsymbol{a},x)}$;	DP2	$\neg\underline{\boldsymbol{A}(x)} \vee \boldsymbol{B}(\boldsymbol{a})$;
CP3	$\neg\boldsymbol{A}(x) \vee \underline{\boldsymbol{R}(\boldsymbol{f}_{\boldsymbol{A}}(x),\boldsymbol{a})}$;	DP3	$\neg\underline{\boldsymbol{B}(y)} \vee \boldsymbol{R}(\boldsymbol{a},y)$;
CP4	$\neg\boldsymbol{A}(x) \vee \underline{\boldsymbol{R}(\boldsymbol{a},\boldsymbol{f}_{\boldsymbol{A}}(x))}$;	DP4	$\neg\boldsymbol{A}(x) \vee \neg\underline{\boldsymbol{B}(\boldsymbol{a})} \vee \boldsymbol{C}(\boldsymbol{f}_{\boldsymbol{A}}(x))$;
CP5	$\underline{\boldsymbol{R}(x,y)}$;	DP5	$\neg\boldsymbol{A}(x) \vee \neg\underline{\boldsymbol{B}(\boldsymbol{f}_{\boldsymbol{A}}(x))} \vee \boldsymbol{C}(\boldsymbol{a})$;
CP6	$\neg\boldsymbol{A}(x) \vee \underline{\boldsymbol{R}(\boldsymbol{f}_{\boldsymbol{A}}(x),y)}$;	DP6	$\neg\underline{\boldsymbol{A}(x)} \vee \boldsymbol{B}(y)$;
CP7	$\neg\boldsymbol{A}(y) \vee \underline{\boldsymbol{R}(x,\boldsymbol{f}_{\boldsymbol{A}}(y))}$;	DP7	$\neg\underline{\boldsymbol{R}(x,y)} \vee \boldsymbol{A}(x)$;
CP8	$\neg\boldsymbol{A}(x) \vee \neg\boldsymbol{B}(y) \vee \underline{\boldsymbol{R}(\boldsymbol{f}_{\boldsymbol{A}}(x),\boldsymbol{g}_{\boldsymbol{B}}(y))}$;	DP8	$\neg\underline{\boldsymbol{R}(x,y)} \vee \boldsymbol{B}(y)$;
		DP9	$\neg\underline{\boldsymbol{B}(y)} \vee \boldsymbol{R}(x,y)$;
		DP10	$\neg\underline{\boldsymbol{A}(x)} \vee \neg\boldsymbol{B}(y) \vee \boldsymbol{R}(x,y)$;
		DP11	$\neg\boldsymbol{A}(x) \vee \neg\underline{\boldsymbol{B}(y)} \vee \boldsymbol{C}(\boldsymbol{f}_{\boldsymbol{A}}(x))$;
		DP12	$\neg\boldsymbol{A}(x) \vee \neg\underline{\boldsymbol{B}(\boldsymbol{f}_{\boldsymbol{A}}(x))} \vee \boldsymbol{C}(y)$;
		DP13	$\neg\boldsymbol{A}(x) \vee \neg\underline{\boldsymbol{B}(y)} \vee \boldsymbol{R}(\boldsymbol{f}_{\boldsymbol{A}}(x),y)$;
		DP14	$\neg\boldsymbol{A}(x) \vee \neg\boldsymbol{B}(y) \vee \neg\underline{\boldsymbol{C}(\boldsymbol{f}_{\boldsymbol{A}}(x))} \vee \boldsymbol{D}(\boldsymbol{g}_{\boldsymbol{B}}(y))$;

Table A.6: Summary of inferences for cross-products of concepts in $\mathcal{EL}$

AP1.	OR[C1; DP1] :	$\underline{A(a)};\ \neg\underline{A(a)} \vee B(x)\ \vdash\ B(x)$	: C3
AP2.	OR[C1; DP2] :	$\underline{A(a)};\ \neg\underline{A(x)} \vee B(b)\ \vdash\ B(b)$	: C1
AP3.	OR[C1; DP3] :	$\underline{A(a)};\ \neg\underline{A(y)} \vee R(b,y)\ \vdash\ R(b,a)$	: C2
AP4.	OR[C1; DP4] :	$\underline{B(a)};\ \neg A(x) \vee \neg\underline{B(a)} \vee C(f_A(x))\ \vdash\ \neg A(x) \vee C(f_A(x))$	: C4

Continued on next page

AP5.	OR[C1; DP6] :	$\underline{A(a)};\ \neg\underline{A(x)} \vee B(y)\ \vdash\ B(x)$	: C3
AP6.	OR[C1; DP9] :	$\underline{A(a)};\ \neg\underline{A(y)} \vee R(x,y)\ \vdash\ R(x,a)$	: CP1
AP7.	OR[C1; DP10] :	$\underline{A(a)};\ \neg\underline{A(x)} \vee \neg B(y) \vee R(x,y)\ \vdash\ \neg B(y) \vee R(a,y)$	: DP3
AP8.	OR[C1; DP11] :	$\underline{B(a)};\ \neg A(x) \vee \neg\underline{B(y)} \vee C(f_A(x))\ \vdash\ \neg A(x) \vee C(f_A(x))$	: C4
AP9.	OR[C1; DP13] :	$\underline{B(a)};\ \neg A(x) \vee \neg\underline{B(y)} \vee R(f_A(x),y)\ \vdash\ \neg A(x) \vee R(f_A(x),a)$	: CP3
AP10.	OR[C2; DP7] :	$\underline{R(a,b)};\ \neg\underline{R(x,y)} \vee A(x)\ \vdash\ A(a)$	: C1
AP11.	OR[C2; DP8] :	$\underline{R(a,b)};\ \neg\underline{R(x,y)} \vee A(y)\ \vdash\ A(b)$	: C1
AP12.	OR[CP1; D2] :	$\underline{R(x,a)};\ \neg\underline{R(b,a)}\ \vdash\ \square$	: $\bot$
AP13.	OR[CP1; D6] :	$\underline{R(x,a)};\ \neg\underline{R(x,y)} \vee \neg B(y) \vee A(x)\ \vdash\ \neg B(a) \vee A(x)$	: DP1
AP14.	OR[CP1; DP7] :	$\underline{R(x,a)};\ \neg\underline{R(x,y)} \vee A(x)\ \vdash\ A(x)$	: C3
AP15.	OR[CP1; DP8] :	$\underline{R(x,a)};\ \neg\underline{R(x,y)} \vee B(y)\ \vdash\ B(a)$	: C1
AP16.	OR[CP2; D2] :	$\underline{R(a,x)};\ \neg\underline{R(a,b)}\ \vdash\ \square$	: $\bot$
AP17.	OR[CP2; D6] :	$\underline{R(a,x)};\ \neg\underline{R(x,y)} \vee \neg B(y) \vee A(x)\ \vdash\ \neg B(x) \vee A(a)$	: DP2
AP18.	OR[CP2; DP7] :	$\underline{R(a,x)};\ \neg\underline{R(x,y)} \vee A(x)\ \vdash\ A(a)$	: C1
AP19.	OR[CP2; DP8] :	$\underline{R(a,x)};\ \neg\underline{R(x,y)} \vee B(y)\ \vdash\ B(x)$	: C3
AP20.	OR[CP3; D6] :	$\neg A(x) \vee \underline{R(f_A(x),a)};\ \neg\underline{R(x,y)} \vee \neg B(y) \vee C(x)\ \vdash$ $\vdash\ \neg A(x) \vee \neg B(a) \vee C(f_A(x))$	: DP4
AP21.	OR[CP3; DP7] :	$\neg A(x) \vee \underline{R(f_A(x),a)};\ \neg\underline{R(x,y)} \vee B(x)\ \vdash\ \neg A(x) \vee B(f_A(x))$	: C4
AP22.	OR[CP3; DP8] :	$\neg A(x) \vee \underline{R(f_A(x),a)};\ \neg\underline{R(x,y)} \vee B(y)\ \vdash\ \neg A(x) \vee B(a)$	: DP2
AP23.	OR[CP4; D6] :	$\neg A(x) \vee \underline{R(a,f_A(x))};\ \neg\underline{R(x,y)} \vee \neg B(y) \vee C(x)\ \vdash$ $\vdash\ \neg A(x) \vee \neg B(f_A(x)) \vee C(a)$	: DP5
AP24.	OR[CP4; DP7] :	$\neg A(x) \vee \underline{R(a,f_A(x))};\ \neg\underline{R(x,y)} \vee B(x)\ \vdash\ \neg A(x) \vee B(a)$	: DP2
AP25.	OR[CP4; DP8] :	$\neg A(x) \vee \underline{R(a,f_A(x))};\ \neg\underline{R(x,y)} \vee B(y)\ \vdash\ \neg A(x) \vee B(f_A(x))$	: C4
AP26.	OR[C3; DP1] :	$\underline{A(x)};\ \neg\underline{A(a)} \vee B(x)\ \vdash\ B(x)$	: C3
AP27.	OR[C3; DP2] :	$\underline{A(x)};\ \neg\underline{A(x)} \vee B(a)\ \vdash\ B(a)$	: C1
AP28.	OR[C3; DP3] :	$\underline{A(x)};\ \neg\underline{A(y)} \vee R(a,y)\ \vdash\ R(a,x)$	: CP2
AP29.	OR[C3; DP4] :	$\underline{B(x)};\ \neg A(x) \vee \neg\underline{B(a)} \vee C(f_A(x))\ \vdash\ \neg A(x) \vee C(f_A(x))$	: C4
AP30.	OR[C3; DP5] :	$\underline{B(x)};\ \neg A(x) \vee \neg\underline{B(f_A(x))} \vee C(a)\ \vdash\ \neg A(x) \vee C(a)$	: DP2
AP31.	OR[C4; DP2] :	$\neg A(x) \vee \underline{B(f_A(x))};\ \neg\underline{B(x)} \vee C(a)\ \vdash\ \neg A(x) \vee C(a)$	: DP2
AP32.	OR[C4; DP3] :	$\neg A(x) \vee \underline{B(f_A(x))};\ \neg\underline{B(y)} \vee R(a,y)\ \vdash\ \neg A(x) \vee R(a,f_A(x))$	: CP4
AP33.	OR[C4; DP5] :	$\neg A(x) \vee \underline{B(f_A(x))};\ \neg A(x) \vee \neg\underline{B(f_A(x))} \vee C(a)\ \vdash$ $\vdash\ \neg A(x) \vee \neg A(x) \vee C(a)\ \vdash\ \neg A(x) \vee C(a)$	: DP2
AP34.	OR[CP5; D2] :	$\underline{R(x,y)};\ \neg\underline{R(a,b)}\ \vdash\ \square$	: $\bot$
TP1.	OR[C3; DP6] :	$\underline{A(x)};\ \neg\underline{A(x)} \vee B(y)\ \vdash\ B(x)$	: C3
TP2.	OR[C3; DP9] :	$\underline{A(x)};\ \neg\underline{B(y)} \vee R(x,y)\ \vdash\ R(x,y)$	: CP5

Continued on next page

TP3.	OR[C3; DP10] :	$\underline{A(x)};\ \neg\underline{A(x)} \vee \neg B(y) \vee R(x,y)\ \vdash\ \neg B(y) \vee R(x,y)$	: DP9
TP4.	OR[C3; DP11] :	$\underline{B(x)};\ \neg A(x) \vee \neg\underline{B(y)} \vee C(f_A(x))\ \vdash\ \neg A(x) \vee C(f_A(x))$	: C4
TP5.	OR[C3; DP12] :	$\underline{B(x)};\ \neg A(x) \vee \neg\underline{B(f_A(x))} \vee C(y)\ \vdash\ \neg A(x) \vee C(y)$	: DP6
TP6.	OR[C3; DP13] :	$\underline{B(x)};\ \neg A(x) \vee \neg\underline{B(y)} \vee R(f_A(x),y)\ \vdash\ \neg A(x) \vee R(f_A(x),y)$	: CP6
TP7.	OR[C3; DP14] :	$\underline{C(x)};\ \neg A(x) \vee \neg B(y) \vee \neg\underline{C(f_A(x))} \vee D(g_B(y))\ \vdash$ $\vdash\ \neg B(x) \vee \neg A(y) \vee D(g_B(x))$	: DP11
TP8.	OR[C4; DP6] :	$\neg A(x) \vee \underline{B(f_A(x))};\ \neg\underline{B(x)} \vee C(y)\ \vdash\ \neg A(x) \vee C(y)$	: DP6
TP9.	OR[C4; DP9] :	$\neg A(x) \vee \underline{B(f_A(x))};\ \neg\underline{B(y)} \vee R(x,y)\ \vdash\ \neg A(y) \vee R(x,f_A(y))$	: CP7
TP10.	OR[C4; DP10] :	$\neg A(x) \vee \underline{B(f_A(x))};\ \neg\underline{B(x)} \vee \neg C(y) \vee R(x,y)\ \vdash$ $\vdash\ \neg A(x) \vee \neg C(y) \vee R(f_A(x),y)$	: DP13
TP11.	OR[C4; DP11] :	$\neg A(x) \vee \underline{C(f_A(x))};\ \neg B(x) \vee \neg\underline{C(y)} \vee D(g_B(x))\ \vdash$ $\vdash\ \neg B(x) \vee \neg A(y) \vee D(g_B(x))$	: DP11
TP12.	OR[C4; DP12] :	$\neg A(x) \vee \underline{B(f_A(x))};\ \neg A(x) \vee \neg\underline{B(f_A(x))} \vee C(y)\ \vdash$ $\vdash\ \neg A(x) \vee \neg A(x) \vee C(y) \vdash \neg A(x) \vee C(y)$	: DP6
TP13.	OR[C4; DP13] :	$\neg A(x) \vee \underline{C(f_A(x))};\ \neg B(x) \vee \neg\underline{C(y)} \vee R(g_B(x),y)\ \vdash$ $\vdash\ \neg B(x) \vee \neg A(y) \vee R(f_B(x),g_A(y))$	: CP8
TP14.	OR[C4; DP14] :	$\neg A(x) \vee \underline{C(f_A(x))};\ \neg A(x) \vee \neg B(y) \vee \neg\underline{C(f_A(x))} \vee D(g_B(y))\ \vdash$ $\vdash\ \neg A(y) \vee \neg B(x) \vee \neg A(y) \vee D(g_B(x))\ \vdash \neg B(x) \vee \neg A(y) \vee D(g_B(x))$	: DP11
TP15.	OR[C5; DP7] :	$\neg A(x) \vee \underline{R(x,f_A(x))};\ \neg\underline{R(x,y)} \vee B(x)\ \vdash\ \neg A(x) \vee B(x)$	: D4
TP16.	OR[C5; DP8] :	$\neg A(x) \vee \underline{R(x,f_A(x))};\ \neg\underline{R(x,y)} \vee B(y)\ \vdash\ \neg A(x) \vee B(f_A(x))$	: C4
TP17.	OR[CP5; D6] :	$\underline{R(x,y)};\ \neg\underline{R(x,y)} \vee \neg B(y) \vee A(x)\ \vdash\ \neg B(x) \vee A(y)$	: DP6
TP18.	OR[CP5; DP7] :	$\underline{R(x,y)};\ \neg\underline{R(x,y)} \vee A(x)\ \vdash\ A(x)$	: C3
TP19.	OR[CP5; DP8] :	$\underline{R(x,y)};\ \neg\underline{R(x,y)} \vee B(y)\ \vdash\ B(x)$	: C3
TP20.	OR[CP6; D6] :	$\neg A(x) \vee \underline{R(f_A(x),y)};\ \neg\underline{R(x,y)} \vee \neg B(y) \vee C(x)\ \vdash$ $\vdash\ \neg A(x) \vee \neg B(y) \vee C(f_A(x))$	: DP11
TP21.	OR[CP6; DP7] :	$\neg A(x) \vee \underline{R(f_A(x),y)};\ \neg\underline{R(x,y)} \vee B(x)\ \vdash\ \neg A(x) \vee B(f_A(x))$	: C4
TP22.	OR[CP6; DP8] :	$\neg A(x) \vee \underline{R(f_A(x),y)};\ \neg\underline{R(x,y)} \vee B(y)\ \vdash\ \neg A(x) \vee B(y)$	: DP6
TP23.	OR[CP7; D6] :	$\neg A(y) \vee \underline{R(x,f_A(y))};\ \neg\underline{R(x,y)} \vee \neg B(y) \vee C(x)\ \vdash$ $\vdash\ \neg A(x) \vee \neg B(f_A(x)) \vee C(y)$	: DP12
TP24.	OR[CP7; DP7] :	$\neg A(y) \vee \underline{R(x,f_A(y))};\ \neg\underline{R(x,y)} \vee B(x)\ \vdash\ \neg A(x) \vee B(y)$	: DP6
TP25.	OR[CP7; DP8] :	$\neg A(y) \vee \underline{R(x,f_A(y))};\ \neg\underline{R(x,y)} \vee B(y)\ \vdash\ \neg A(x) \vee B(f_A(x))$	: C4
TP26.	OR[CP8; D6] :	$\neg A(x) \vee \neg B(y) \vee \underline{R(f_A(x),g_B(y))};\ \neg\underline{R(x,y)} \vee \neg C(y) \vee D(x)\ \vdash$ $\vdash\ \neg B(x) \vee \neg A(y) \vee \neg C(g_B(x)) \vee D(f_A(y))$	: DP14
TP27.	OR[CP8; DP7] :	$\neg A(x) \vee \neg B(y) \vee \underline{R(f_A(x),g_B(y))};\ \neg\underline{R(x,y)} \vee C(x)\ \vdash$ $\vdash\ \neg A(x) \vee \neg B(y) \vee C(f_A(x))$	: DP11
TP28.	OR[CP8; DP8] :	$\neg A(x) \vee \neg B(y) \vee \underline{R(f_A(x),g_B(y))};\ \neg\underline{R(x,y)} \vee C(y)\ \vdash$ $\vdash\ \neg B(x) \vee \neg A(y) \vee C(g_B(x))$	: DP11

Continued on next page

AHP1.	$\texttt{OR}[\mathsf{CP1};\mathsf{DH1}] : \underline{R(x,a)};\ \neg\underline{R(x,y)} \vee S(x,y) \ \vdash\ S(x,a)$	$: \mathsf{CP1}$
AHP2.	$\texttt{OR}[\mathsf{CP2};\mathsf{DH1}] : \underline{R(a,x)};\ \neg\underline{R(x,y)} \vee S(x,y) \ \vdash\ S(a,x)$	$: \mathsf{CP2}$
AHP3.	$\texttt{OR}[\mathsf{CP3};\mathsf{DH1}] : \neg A(x) \vee \underline{R(f_A(x),a)};\ \neg\underline{R(x,y)} \vee S(x,y) \ \vdash\ \neg A(x) \vee S(f_A(x),a)$	$: \mathsf{CP3}$
AHP4.	$\texttt{OR}[\mathsf{CP4};\mathsf{DH1}] : \neg A(x) \vee \underline{R(a,f_A(x))};\ \neg\underline{R(x,y)} \vee S(x,y) \ \vdash\ \neg A(x) \vee S(a,f_A(x))$	$: \mathsf{CP4}$
THP1.	$\texttt{OR}[\mathsf{CP5};\mathsf{DH1}] : \underline{R(x,y)};\ \neg\underline{R(x,y)} \vee S(x,y) \ \vdash\ S(x,y)$	$: \mathsf{CP5}$
THP2.	$\texttt{OR}[\mathsf{CP6};\mathsf{DH1}] : \neg A(x) \vee \underline{R(f_A(x),y)};\ \neg\underline{R(x,y)} \vee S(x,y) \ \vdash\ \neg A(x) \vee S(f_A(x),y)$	$: \mathsf{CP6}$
THP3.	$\texttt{OR}[\mathsf{CP7};\mathsf{DH1}] : \neg A(y) \vee \underline{R(x,f_A(y))};\ \neg\underline{R(x,y)} \vee S(x,y) \ \vdash\ \neg A(y) \vee S(x,f_A(y))$	$: \mathsf{CP7}$
THP4.	$\texttt{OR}[\mathsf{CP8};\mathsf{DH1}] : \neg A(x) \vee \neg B(y) \vee \underline{R(f_A(x),g_B(y))};\ \neg\underline{R(x,y)} \vee S(x,y) \ \vdash$ $\vdash\ \neg A(x) \vee \neg B(y) \vee S(f_A(x),g_B(y))$	$: \mathsf{CP8}$

Table A.7: An extension of the datalog program for reasoning with cross-products of concepts in $\mathcal{EL}$

AP1.	$\mathsf{C3}(B)$	$\leftarrow \mathsf{C1}(A,a),\ \mathsf{DP1}(A,B,a);$
AP2.	$\mathsf{C1}(B,b)$	$\leftarrow \mathsf{C1}(A,a),\ \mathsf{DP2}(A,B,b);$
AP3.	$\mathsf{C2}(R,b,a)$	$\leftarrow \mathsf{C1}(A,a),\ \underline{\mathsf{DP3}(A,R,b)};$
AP4.	$\mathsf{C4}(A,C,f_A)$	$\leftarrow \mathsf{C1}(B,a),\ \mathsf{DP4}(A,B,C,f_A,a);$
AP5.	$\mathsf{CP1}(R,a)$	$\leftarrow \mathsf{C1}(A,a),\ \underline{\mathsf{DP9}(A,R)};$
AP6.	$\mathsf{CP1}(R,a)$	$\leftarrow \mathsf{C1}(A,a),\ \underline{\mathsf{DP9}(A,R)};$
AP7.	$\mathsf{DP3}(B,R,a)$	$\leftarrow \mathsf{C1}(A,a),\ \underline{\mathsf{DP10}(A,B,R)};$
AP8.	$\mathsf{C4}(A,C,f_A)$	$\leftarrow \mathsf{C1}(B,a),\ \mathsf{DP11}(A,B,C,f_A);$
AP9.	$\mathsf{CP3}(A,R,f_A,a)$	$\leftarrow \mathsf{C1}(B,a),\ \underline{\mathsf{DP13}(A,B,R,f_A)};$
AP10.	$\mathsf{C1}(A,a)$	$\leftarrow \mathsf{C2}(R,a,b),\ \underline{\mathsf{DP7}(R,A)};$
AP11.	$\mathsf{C1}(A,b)$	$\leftarrow \mathsf{C2}(R,a,b),\ \underline{\mathsf{DP8}(R,A)};$
AP12.	$\bot$	$\leftarrow \mathsf{CP1}(R,a),\ \underline{\mathsf{D2}(R,b,a)};$
AP13.	$\mathsf{DP1}(B,A,a)$	$\leftarrow \mathsf{CP1}(R,a),\ \underline{\mathsf{D6}(R,B,A)};$
AP14.	$\mathsf{C3}(A)$	$\leftarrow \mathsf{CP1}(R,a),\ \underline{\mathsf{DP7}(R,A)};$
AP15.	$\mathsf{C1}(B,a)$	$\leftarrow \mathsf{CP1}(R,a),\ \underline{\mathsf{DP8}(R,B)};$
AP16.	$\bot$	$\leftarrow \mathsf{CP2}(R,a),\ \underline{\mathsf{D2}(R,a,b)};$
AP17.	$\mathsf{DP2}(B,A,a)$	$\leftarrow \mathsf{CP2}(R,a),\ \underline{\mathsf{D6}(R,B,A)};$
AP18.	$\mathsf{C1}(A,a)$	$\leftarrow \mathsf{CP2}(R,a),\ \underline{\mathsf{DP7}(R,A)};$
AP19.	$\mathsf{C3}(B)$	$\leftarrow \mathsf{CP2}(R,a),\ \underline{\mathsf{DP8}(R,B)};$
AP20.	$\mathsf{DP4}(A,B,C,f_A,a)$	$\leftarrow \mathsf{CP3}(A,R,f_A,a),\ \underline{\mathsf{D6}(R,B,C)};$
AP21.	$\mathsf{C4}(A,B,f_A)$	$\leftarrow \mathsf{CP3}(A,R,f_A,a),\ \underline{\mathsf{DP7}(R,B)};$
AP22.	$\mathsf{DP2}(A,B,a)$	$\leftarrow \mathsf{CP3}(A,R,f_A,a),\ \underline{\mathsf{DP8}(R,B)};$
AP23.	$\mathsf{DP5}(A,B,C,f_A,a)$	$\leftarrow \mathsf{CP4}(A,R,f_A,a),\ \underline{\mathsf{D6}(R,B,C)};$
AP24.	$\mathsf{DP2}(A,B,a)$	$\leftarrow \mathsf{CP4}(A,R,f_A,a),\ \underline{\mathsf{DP7}(R,B)};$
AP25.	$\mathsf{C4}(A,B,f_A)$	$\leftarrow \mathsf{CP4}(A,R,f_A,a),\ \underline{\mathsf{DP8}(R,B)};$
AP26.	$\mathsf{C3}(B)$	$\leftarrow \mathsf{C3}(A),\ \mathsf{DP1}(A,B,a);$
AP27.	$\mathsf{C1}(B,a)$	$\leftarrow \mathsf{C3}(A),\ \mathsf{DP2}(A,B,a);$

Continued on next page

AP28.	$\mathsf{CP2}(R,a)$	$\leftarrow \mathsf{C3}(A),\ \dashuline{\mathsf{DP3}(A,R,a)};$
AP29.	$\mathsf{C4}(A,C,f_A)$	$\leftarrow \mathsf{C3}(B),\ \mathsf{DP4}(A,B,C,f_A,a);$
AP30.	$\mathsf{DP2}(A,C,a)$	$\leftarrow \mathsf{C3}(B),\ \mathsf{DP5}(A,B,C,f_A,a);$
AP31.	$\mathsf{DP2}(A,C,a)$	$\leftarrow \mathsf{C4}(A,B,f_A),\ \mathsf{DP2}(B,C,a);$
AP32.	$\mathsf{CP4}(A,R,f_A,a)$	$\leftarrow \mathsf{C4}(A,B,f_A),\ \dashuline{\mathsf{DP3}(B,R,a)};$
AP33.	$\mathsf{DP2}(A,C,a)$	$\leftarrow \mathsf{C4}(A,B,f_A),\ \mathsf{DP5}(A,B,C,f_A,a);$
AP34.	$\bot$	$\leftarrow \mathsf{CP5}(R),\ \underline{\mathsf{D2}(R,a,b)};$
TP1.	$\mathsf{C3}(B)$	$\leftarrow \mathsf{C3}(A),\ \mathsf{DP6}(A,B);$
TP2.	$\mathsf{CP5}(R)$	$\leftarrow \mathsf{C3}(A),\ \underline{\mathsf{DP9}(B,R)};$
TP3.	$\mathsf{DP9}(B,R)$	$\leftarrow \mathsf{C3}(A),\ \underline{\mathsf{DP10}(A,B,R)};$
TP4.	$\mathsf{C4}(A,C,f_A)$	$\leftarrow \mathsf{C3}(B),\ \mathsf{DP11}(A,B,C,f_A);$
TP5.	$\mathsf{DP6}(A,C)$	$\leftarrow \mathsf{C3}(B),\ \mathsf{DP12}(A,B,C,f_A);$
TP6.	$\mathsf{CP6}(A,R,f_A)$	$\leftarrow \mathsf{C3}(B),\ \dashuline{\mathsf{DP13}(A,B,R,f_A)};$
TP7.	$\mathsf{DP11}(B,A,D,g_B)$	$\leftarrow \mathsf{C3}(C),\ \mathsf{DP14}(A,B,C,D,f_A,g_B);$
TP8.	$\mathsf{DP6}(A,C)$	$\leftarrow \mathsf{C4}(A,B,f_A),\ \mathsf{DP6}(B,C);$
TP9.	$\mathsf{CP7}(A,R,f_A)$	$\leftarrow \mathsf{C4}(A,B,f_A),\ \underline{\mathsf{DP9}(B,R)};$
TP10.	$\mathsf{DP13}(A,C,R,f_A)$	$\leftarrow \mathsf{C4}(A,B,f_A),\ \underline{\mathsf{DP10}(B,C,R)};$
TP11.	$\mathsf{DP11}(B,A,D,g_B)$	$\leftarrow \mathsf{C4}(A,C,f_A),\ \mathsf{DP11}(B,C,D,g_B);$
TP12.	$\mathsf{DP6}(A,C)$	$\leftarrow \mathsf{C4}(A,B,f_A),\ \mathsf{DP12}(A,B,C,f_A);$
TP13.	$\mathsf{CP8}(B,A,R,f_A,g_B)$	$\leftarrow \mathsf{C4}(A,C,f_A),\ \dashuline{\mathsf{DP13}(B,C,R,g_B)};$
TP14.	$\mathsf{DP11}(B,A,D,g_B)$	$\leftarrow \mathsf{C4}(A,C,f_A),\ \mathsf{DP14}(A,B,C,D,f_A,g_B);$
TP15.	$\mathsf{D4}(A,B)$	$\leftarrow \mathsf{C5}(A,R,f_A),\ \underline{\mathsf{DP7}(R,B)};$
TP16.	$\mathsf{C4}(A,B,f_A)$	$\leftarrow \mathsf{C5}(A,R,f_A),\ \underline{\mathsf{DP8}(R,B)};$
TP17.	$\mathsf{DP6}(B,A)$	$\leftarrow \mathsf{CP5}(R),\ \underline{\mathsf{D6}(R,B,A)};$
TP18.	$\mathsf{C3}(A)$	$\leftarrow \mathsf{CP5}(R),\ \underline{\mathsf{DP7}(R,A)};$
TP19.	$\mathsf{C3}(B)$	$\leftarrow \mathsf{CP5}(R),\ \underline{\mathsf{DP8}(R,B)};$
TP20.	$\mathsf{DP11}(A,B,C,f_A)$	$\leftarrow \mathsf{CP6}(A,R,f_A),\ \underline{\mathsf{D6}(R,B,C)};$
TP21.	$\mathsf{C4}(A,B,f_A)$	$\leftarrow \mathsf{CP6}(A,R,f_A),\ \underline{\mathsf{DP7}(R,B)};$
TP22.	$\mathsf{DP6}(A,B)$	$\leftarrow \mathsf{CP6}(A,R,f_A),\ \underline{\mathsf{DP8}(R,B)};$
TP23.	$\mathsf{DP12}(A,B,C,f_A)$	$\leftarrow \mathsf{CP7}(A,R,f_A),\ \underline{\mathsf{D6}(R,B,C)};$
TP24.	$\mathsf{DP6}(A,B)$	$\leftarrow \mathsf{CP7}(A,R,f_A),\ \underline{\mathsf{DP7}(R,B)};$
TP25.	$\mathsf{C4}(A,B,f_A)$	$\leftarrow \mathsf{CP7}(A,R,f_A),\ \underline{\mathsf{DP8}(R,B)};$
TP26.	$\mathsf{DP14}(B,A,C,D,g_B,f_A)$	$\leftarrow \mathsf{CP8}(A,B,R,f_A,g_B),\ \underline{\mathsf{D6}(R,C,D)};$
TP27.	$\mathsf{DP11}(A,B,C,f_A)$	$\leftarrow \mathsf{CP8}(A,B,R,f_A,g_B),\ \underline{\mathsf{DP7}(R,C)};$
TP28.	$\mathsf{DP11}(B,A,C,g_B)$	$\leftarrow \mathsf{CP8}(A,B,R,f_A,g_B),\ \underline{\mathsf{DP8}(R,C)};$
AHP1.	$\mathsf{CP1}(S,a)$	$\leftarrow \mathsf{CP1}(R,a),\ \underline{\mathsf{DH1}(R,S)};$
AHP2.	$\mathsf{CP2}(S,a)$	$\leftarrow \mathsf{CP2}(R,a),\ \underline{\mathsf{DH1}(R,S)};$
AHP3.	$\mathsf{CP3}(A,S,f,a)$	$\leftarrow \mathsf{CP3}(A,R,f,a),\ \underline{\mathsf{DH1}(R,S)};$
AHP4.	$\mathsf{CP4}(A,S,f,a)$	$\leftarrow \mathsf{CP4}(A,R,f,a),\ \underline{\mathsf{DH1}(R,S)};$
THP1.	$\mathsf{CP5}(S)$	$\leftarrow \mathsf{CP5}(R),\ \underline{\mathsf{DH1}(R,S)};$
THP2.	$\mathsf{CP6}(A,S,f)$	$\leftarrow \mathsf{CP6}(A,R,f),\ \underline{\mathsf{DH1}(R,S)};$
THP3.	$\mathsf{CP7}(A,S,f)$	$\leftarrow \mathsf{CP7}(A,R,f),\ \underline{\mathsf{DH1}(R,S)};$

Continued on next page

THP4. CP8(A, B, S, f, g)	$\leftarrow$ CP8(A, B, R, f, g), $\underline{\text{DH1}(R, S)}$;

A.5 Extensions of DL $\mathcal{EL}$ with Nominals

In this appendix we provide additional details about the paramodulation-based decision procedure for extensions of $\mathcal{EL}$ with nominals considered in subsection 2.4.3. We demonstrate that all new clauses that can be derived from the input clauses for $\mathcal{EL}$ with nominals are of the types given in Table A.8.

Table A.8 Clause types for extensions of $\mathcal{EL}$ with n**o**minals

CO1	$\underline{\boldsymbol{a}} \simeq \boldsymbol{b}$;	DHXO1	$\neg\underline{\boldsymbol{S}(x, \boldsymbol{a})} \vee \boldsymbol{T}(x, \boldsymbol{b})$;
CO2	$\underline{x} \simeq o$;	DHXO2	$\neg\underline{\boldsymbol{A}(x)} \vee \neg\boldsymbol{S}(x, \boldsymbol{a}) \vee \boldsymbol{T}(x, \boldsymbol{b})$;
CO3	$\underline{\boldsymbol{R}(x, \boldsymbol{a})}$;	DHXO3	$\neg\boldsymbol{A}(x) \vee \neg\underline{\boldsymbol{S}(\boldsymbol{a}, \boldsymbol{b})} \vee \boldsymbol{T}(\boldsymbol{f}_{\boldsymbol{A}}(x), \boldsymbol{c})$;
CO4	$\neg\boldsymbol{A}(x) \vee \underline{\boldsymbol{f}_{\boldsymbol{A}}(x)} \simeq \boldsymbol{a}$;	DHXO4	$\neg\boldsymbol{A}(x) \vee \neg\underline{\boldsymbol{S}(\boldsymbol{f}_{\boldsymbol{A}}(x), \boldsymbol{a})} \vee \boldsymbol{T}(\boldsymbol{f}_{\boldsymbol{A}}(x), \boldsymbol{b})$;
CO5	$\neg\boldsymbol{A}(x) \vee \underline{\boldsymbol{R}(\boldsymbol{f}_{\boldsymbol{A}}(x), \boldsymbol{a})}$;	DPO1	$\neg\boldsymbol{A}(x) \vee \neg\underline{\boldsymbol{B}(\boldsymbol{a})} \vee \boldsymbol{C}(y)$;
DO1	$\neg\underline{\boldsymbol{A}(x)} \vee \boldsymbol{a} \simeq \boldsymbol{b}$;	DPO2	$\neg\underline{\boldsymbol{A}(x)} \vee \boldsymbol{R}(\boldsymbol{a}, y)$
DO2	$\neg\underline{\boldsymbol{A}(x)} \vee x \simeq \boldsymbol{a}$;	DPO3	$\neg\underline{\boldsymbol{A}(x)} \vee \boldsymbol{R}(y, \boldsymbol{a})$
DO3	$\neg\underline{\boldsymbol{A}(\boldsymbol{a})} \vee \boldsymbol{B}(x)$;	DPO4	$\neg\boldsymbol{A}(x) \vee \neg\underline{\boldsymbol{B}(y)} \vee \boldsymbol{R}(\boldsymbol{f}_{\boldsymbol{A}}(x), \boldsymbol{a})$;
DO4	$\neg\underline{\boldsymbol{A}(x)} \vee \boldsymbol{B}(\boldsymbol{a})$;	DPO5	$\neg\boldsymbol{A}(x) \vee \neg\underline{\boldsymbol{B}(y)} \vee \boldsymbol{R}(\boldsymbol{a}, \boldsymbol{f}_{\boldsymbol{A}}(x))$;
DO5	$\neg\underline{\boldsymbol{A}(x)} \vee \boldsymbol{R}(\boldsymbol{a}, \boldsymbol{b})$;	DPO6	$\neg\boldsymbol{A}(x) \vee \neg\boldsymbol{B}(y) \vee \neg\underline{\boldsymbol{C}(\boldsymbol{a})} \vee \boldsymbol{D}(\boldsymbol{f}_{\boldsymbol{A}}(x))$;
DO6	$\neg\underline{\boldsymbol{A}(x)} \vee \boldsymbol{R}(x, \boldsymbol{a})$;		
DO7	$\neg\boldsymbol{A}(x) \vee \neg\underline{\boldsymbol{B}(\boldsymbol{a})} \vee \boldsymbol{C}(\boldsymbol{b})$;		
DO8	$\neg\boldsymbol{A}(x) \vee \neg\underline{\boldsymbol{B}(\boldsymbol{a})} \vee \boldsymbol{C}(x)$;		
DO9	$\neg\boldsymbol{A}(x) \vee \neg\underline{\boldsymbol{B}(\boldsymbol{a})} \vee \boldsymbol{C}(\boldsymbol{f}_{\boldsymbol{A}}(x))$;		

Clause types CO1 – CO5 and DO1 – DO9 correspond to the extension of $\mathcal{EL}$ with nominals. Simple role hierarchies do not give any new clause types, whereas role conjunctions or cross-products of concepts give new clause types DHXO1 – DHXO4 and DPO1 – DPO6 respectively (recall that we cannot use both of these constructors since the resulted logic is already intractable without nominals).

In Table A.9 we have listed all possible inferences between clause types for $\mathcal{EL}$ and its extensions with cross-products and (extended) role hierarchies that we have derived before, and new clause types for nominals given in Table A.8. We have used the standard paramodulation calculus described by rules from Figure 2.1 and Figure 2.2, except for couple of modifications that we did to simplify an (already large) case analysis of possible inferences.

First, we assume that every clause of form $x \simeq a$ is immediately simplified to $x \simeq o$, where o is a special constant that is *least* in the ordering (see inference AO11). This simplification is correct, since clause $x \simeq a$ semantically means that the underlying model consists of one element. The simplification is done to avoid paramodulation inferences *from* variable a, which otherwise were possible. Second, in inferences AHXO14 and AHXO25 we use *simultaneous paramodulation* into terms $f_A(x)$. This strategy is discussed in 3.5.3 on p. 87. Many other simplification techniques discussed in this thesis can be applied to simplify the case analysis of inferences and the resulted procedure.

A correspondent extensions of a datalog program for reasoning with nominals are summarised in Table A.10.

Table A.9: Summary of inferences for nominals in $\mathcal{EL}$

AO1.	OR[C1; DO1] :	$\underline{A(a)};\ \neg\underline{A(x)} \vee b \simeq c\ \vdash\ b \simeq c$	: CO1
AO2.	OR[C1; DO2] :	$\underline{A(a)};\ \neg\underline{A(x)} \vee x \simeq b\ \vdash\ a \simeq b$	: CO1
AO3.	OR[C1; DO3] :	$\underline{A(a)};\ \neg\underline{A(a)} \vee B(x)\ \vdash\ B(x)$	: C3
AO4.	OR[C1; DO4] :	$\underline{A(a)};\ \neg\underline{A(x)} \vee B(b)\ \vdash\ B(b)$	: C1
AO5.	OR[C1; DO5] :	$\underline{A(a)};\ \neg\underline{A(x)} \vee R(a,b)\ \vdash\ R(a,b)$	: C2
AO6.	OR[C1; DO6] :	$\underline{A(a)};\ \neg\underline{A(x)} \vee R(x,b)\ \vdash\ R(a,b)$	: C2
AO7.	OR[C1; DO7] :	$\underline{B(a)};\ \neg A(x) \vee \neg\underline{B(a)} \vee C(b)\ \vdash\ \neg A(x) \vee C(b)$	: DO4
AO8.	OR[C1; DO8] :	$\underline{B(a)};\ \neg A(x) \vee \neg\underline{B(a)} \vee C(x)\ \vdash\ \neg A(x) \vee C(x)$	: D4
AO9.	OR[C1; DO9] :	$\underline{B(a)};\ \neg A(x) \vee \neg\underline{B(a)} \vee C(f_A(x))\ \vdash\ \neg A(x) \vee C(f_A(x))$	: C4
AO10.	OR[C3; DO1] :	$\underline{A(x)};\ \neg\underline{A(x)} \vee b \simeq c\ \vdash\ b \simeq c$	: CO1
AO11.	OR[C3; DO2] :	$\underline{A(x)};\ \neg\underline{A(x)} \vee x \simeq b\ \vdash\ x \simeq b\ \vdash\ x \simeq o$	: CO2
AO12.	OR[C3; DO3] :	$\underline{A(x)};\ \neg\underline{A(a)} \vee B(x)\ \vdash\ B(x)$	: C3
AO13.	OR[C3; DO4] :	$\underline{A(x)};\ \neg\underline{A(x)} \vee B(b)\ \vdash\ B(b)$	: C1
AO14.	OR[C3; DO5] :	$\underline{A(x)};\ \neg\underline{A(x)} \vee R(a,b)\ \vdash\ R(a,b)$	: C2
AO15.	OR[C3; DO6] :	$\underline{A(x)};\ \neg\underline{A(x)} \vee R(x,a)\ \vdash\ R(x,a)$	: CO3
AO16.	OR[C3; DO7] :	$\underline{B(x)};\ \neg A(x) \vee \neg\underline{B(a)} \vee C(b)\ \vdash\ \neg A(x) \vee C(b)$	: DO4
AO17.	OR[C3; DO8] :	$\underline{B(x)};\ \neg A(x) \vee \neg\underline{B(a)} \vee C(x)\ \vdash\ \neg A(x) \vee C(x)$	: D4
AO18.	OR[C3; DO9] :	$\underline{B(x)};\ \neg A(x) \vee \neg\underline{B(a)} \vee C(f_A(x))\ \vdash\ \neg A(x) \vee C(f_A(x))$	: C4
AO19.	OR[C4; DO1] :	$\neg A(x) \vee \underline{B(f_A(x))};\ \neg\underline{B(x)} \vee b \simeq c\ \vdash\ \neg A(x) \vee b \simeq c$	: DO1
AO20.	OR[C4; DO2] :	$\neg A(x) \vee \underline{B(f_A(x))};\ \neg\underline{B(x)} \vee x \simeq b\ \vdash\ \neg A(x) \vee f_A(x) \simeq b$	: CO4
AO21.	OR[C4; DO4] :	$\neg A(x) \vee \underline{B(f_A(x))};\ \neg\underline{B(x)} \vee C(b)\ \vdash\ \neg A(x) \vee C(b)$	: DO4
AO22.	OR[C4; DO5] :	$\neg A(x) \vee \underline{B(f_A(x))};\ \neg\underline{B(x)} \vee R(a,b)\ \vdash\ \neg A(x) \vee R(a,b)$	: DO5
AO23.	OR[C4; DO6] :	$\neg A(x) \vee \underline{B(f_A(x))};\ \neg\underline{B(x)} \vee R(x,a)\ \vdash\ \neg A(x) \vee R(f_A(x),a)$	: CO5
AO24.	OP[CO1; C1] :	$\underline{a} \simeq b;\ A(\underline{a})\ \vdash\ A(b)$	: C1
AO25.	OP[CO1; C2] :	$\underline{a} \simeq b;\ R(\underline{a},c)\ \vdash\ R(b,c)$	: C2
AO26.	OP[CO1; C2] :	$\underline{a} \simeq b;\ R(c,\underline{a})\ \vdash\ R(c,b)$	: C2
AO27.	OP[CO1; D1] :	$\underline{a} \simeq b;\ \neg A(\underline{a})\ \vdash\ \neg A(b)$	: D1

Continued on next page

AO28.	OP[CO1; D2] :	$\underline{a} \simeq b;\ \neg R(\underline{a}, c)\ \vdash\ \neg R(b, c)$	: D2
AO29.	OP[CO1; D2] :	$\underline{a} \simeq b;\ \neg R(c, \underline{a})\ \vdash\ \neg R(c, b)$	: D2
AO30.	OP[CO1; D3] :	$\underline{a} \simeq b;\ \neg A(\underline{a}) \vee B(c)\ \vdash\ \neg A(b) \vee B(c)$	: D3
AO31.	OP[CO1; CO1] :	$\underline{a} \simeq b;\ \underline{a} \simeq c\ \vdash\ b \simeq c$	: CO1
AO32.	OP[CO1; CO1] :	$\underline{a} \simeq b;\ c \simeq \underline{a}\ \vdash\ c \simeq b$	: CO1
AO33.	OP[CO1; CO3] :	$\underline{a} \simeq b;\ R(x, \underline{a})\ \vdash\ R(x, b)$	: CO3
AO34.	OP[CO1; CO4] :	$\underline{a} \simeq b;\ \neg A(x) \vee f_A(x) \simeq \underline{a}\ \vdash\ \neg A(x) \vee f_A(x) \simeq b$	: CO4
AO35.	OP[CO1; CO5] :	$\underline{a} \simeq b;\ \neg A(x) \vee R(f_A(x), \underline{a})\ \vdash\ \neg A(x) \vee R(f_A(x), b)$	: CO5
AO36.	OP[CO1; DO3] :	$\underline{a} \simeq b;\ \neg A(\underline{a}) \vee B(x)\ \vdash\ \neg A(b) \vee B(x)$	: DO3
AO37.	OP[CO1; DO7] :	$\underline{a} \simeq b;\ \neg A(x) \vee \neg B(\underline{a}) \vee C(c)\ \vdash\ \neg A(x) \vee \neg B(b) \vee C(c)$	: DO7
AO38.	OP[CO1; DO8] :	$\underline{a} \simeq b;\ \neg A(x) \vee \neg B(\underline{a}) \vee C(x)\ \vdash\ \neg A(x) \vee \neg B(b) \vee C(x)$	: DO8
AO39.	OP[CO1; DO9] :	$\underline{a} \simeq b;\ \neg A(x) \vee \neg B(\underline{a}) \vee C(f_A(x))\ \vdash$ $\vdash\ \neg A(x) \vee \neg B(b) \vee C(f_A(x))$	: DO9
AO40.	OP[CO2; C1] :	$\underline{x} \simeq o;\ A(\underline{a})\ \vdash\ A(o)$	: C1
AO41.	OP[CO2; C2] :	$\underline{x} \simeq o;\ R(\underline{a}, c)\ \vdash\ R(o, c)$	: C2
AO42.	OP[CO2; C4] :	$\underline{x} \simeq o;\ \neg A(x) \vee B(\underline{f_A(x)})\ \vdash\ \neg A(x) \vee B(o)$	: DO4
AO43.	OP[CO2; C5] :	$\underline{x} \simeq o;\ \neg A(x) \vee R(x, \underline{f_A(x)})\ \vdash\ \neg A(x) \vee R(x, o)$	: DO6
AO44.	OP[CO2; D1] :	$\underline{x} \simeq o;\ \neg A(\underline{a})\ \vdash\ \neg A(o)$	: D1
AO45.	OP[CO2; D2] :	$\underline{x} \simeq o;\ \neg R(\underline{a}, c)\ \vdash\ \neg R(o, c)$	: D2
AO46.	OP[CO2; D2] :	$\underline{x} \simeq o;\ \neg R(c, \underline{a})\ \vdash\ \neg R(c, o)$	: D2
AO47.	OP[CO2; D3] :	$\underline{x} \simeq o;\ \neg A(\underline{a}) \vee B(c)\ \vdash\ \neg A(o) \vee B(c)$	: D3
AO48.	OP[CO2; D7] :	$\underline{x} \simeq o;\ \neg A(x) \vee \neg B(\underline{f_A(x)}) \vee C(x)\ \vdash\ \neg A(x) \vee \neg B(o) \vee C(x)$	: DO8
AO49.	OP[CO2; D8] :	$\underline{x} \simeq o;\ \neg A(x) \vee \neg B(\underline{f_A(x)}) \vee C(f_A(x))\ \vdash$ $\vdash\ \neg A(x) \vee \neg B(o) \vee C(f_A(x))$	: DO9
AO50.	OP[CO2; CO1] :	$\underline{x} \simeq o;\ \underline{a} \simeq c\ \vdash\ o \simeq c$	: CO1
AO51.	OP[CO2; CO1] :	$\underline{x} \simeq o;\ c \simeq \underline{a}\ \vdash\ c \simeq o$	: CO1
AO52.	OP[CO2; CO3] :	$\underline{x} \simeq o;\ R(x, \underline{a})\ \vdash\ R(x, o)$	: CO3
AO53.	OP[CO2; CO4] :	$\underline{x} \simeq o;\ \neg A(x) \vee \underline{f_A(x)} \simeq c\ \vdash\ \neg A(x) \vee o \simeq c$	: DO1
AO54.	OP[CO2; CO4] :	$\underline{x} \simeq o;\ \neg A(x) \vee f_A(x) \simeq \underline{a}\ \vdash\ \neg A(x) \vee f_A(x) \simeq o$	: CO4
AO55.	OP[CO2; CO5] :	$\underline{x} \simeq o;\ \neg A(x) \vee R(\underline{f_A(x)}, c)\ \vdash\ \neg A(x) \vee R(o, c)$	: DO5
AO56.	OP[CO2; CO5] :	$\underline{x} \simeq o;\ \neg A(x) \vee R(f_A(x), \underline{a})\ \vdash\ \neg A(x) \vee R(f_A(x), o)$	: CO5
AO57.	OP[CO2; DO3] :	$\underline{x} \simeq o;\ \neg A(\underline{a}) \vee B(x)\ \vdash\ \neg A(o) \vee B(x)$	: DO3
AO58.	OP[CO2; DO7] :	$\underline{x} \simeq o;\ \neg A(x) \vee \neg B(\underline{a}) \vee C(c)\ \vdash\ \neg A(x) \vee \neg B(o) \vee C(c)$	: DO7
AO59.	OP[CO2; DO8] :	$\underline{x} \simeq o;\ \neg A(x) \vee \neg B(\underline{a}) \vee C(x)\ \vdash\ \neg A(x) \vee \neg B(o) \vee C(x)$	: DO8
AO60.	OP[CO2; DO9] :	$\underline{x} \simeq o;\ \neg A(x) \vee \neg B(\underline{a}) \vee C(f_A(x))\ \vdash$ $\vdash\ \neg A(x) \vee \neg B(o) \vee C(f_A(x))$	: DO9
AO61.	OR[CO3; D2] :	$\underline{R(x, b)};\ \neg \underline{R(a, b)}\ \vdash\ \square$	: $\bot$

Continued on next page

AO62.	OR[CO3; D6] :	$\underline{R(x,b)};\ \neg\underline{R(x,y)} \lor \neg B(y) \lor A(x)\ \vdash\ \neg B(b) \lor A(x)$	: DO3
AO63.	OP[CO4; C4] :	$\neg A(x) \lor \underline{f_A(x)} \simeq a;\ \neg A(x) \lor B(\underline{f_A(x)})\ \vdash$ $\vdash\ \neg A(x) \lor \neg A(x) \lor B(a)\ \vdash\ \neg A(x) \lor B(a)$	: DO4
AO64.	OP[CO4; C5] :	$\neg A(x) \lor \underline{f_A(x)} \simeq a;\ \neg A(x) \lor R(x, \underline{f_A(x)})\ \vdash$ $\vdash\ \neg A(x) \lor \neg A(x) \lor R(x,a)\ \vdash\ \neg A(x) \lor R(x,a)$	: DO6
AO65.	OP[CO4; D7] :	$\neg A(x) \lor \underline{f_A(x)} \simeq a;\ \neg A(x) \lor \neg B(\underline{f_A(x)}) \lor C(x)\ \vdash$ $\vdash\ \neg A(x) \lor \neg A(x) \lor \neg B(a) \lor C(x)\ \vdash\ \neg A(x) \lor \neg B(a) \lor C(x)$	: DO8
AO66.	OP[CO4; D8] :	$\neg A(x) \lor \underline{f_A(x)} \simeq a;\ \neg A(x) \lor \neg B(\underline{f_A(x)}) \lor C(f_A(x))\ \vdash$ $\vdash\ \neg A(x) \lor \neg A(x) \lor \neg B(a) \lor C(f_A(x))\ \vdash\ \neg A(x) \lor \neg B(a) \lor C(f_A(x))$	: DO9
AO67.	OP[CO4; CO4] :	$\neg A(x) \lor \underline{f_A(x)} \simeq a;\ \neg A(x) \lor \underline{f_A(x)} \simeq b\ \vdash$ $\vdash\ \neg A(x) \lor \neg A(x) \lor a \simeq b\ \vdash\ \neg A(x) \lor a \simeq b$	: DO1
AO68.	OP[CO4; CO5] :	$\neg A(x) \lor \underline{f_A(x)} \simeq a;\ \neg A(x) \lor R(\underline{f_A(x)}, b)\ \vdash$ $\vdash\ \neg A(x) \lor \neg A(x) \lor R(a,b)\ \vdash\ \neg A(x) \lor R(a,b)$	: DO5
AO69.	OR[CO5; D6] :	$\neg A(x) \lor \underline{R(f_A(x),a)};\ \neg\underline{R(x,y)} \lor \neg B(y) \lor C(x)\ \vdash$ $\vdash\ \neg A(x) \lor \neg B(a) \lor C(f_A(x))$	: DO9
AHO1.	OR[CO3; DH1] :	$\underline{R(x,a)};\ \neg\underline{R(x,y)} \lor S(x,y)\ \vdash\ S(x,a)$	: CO3
AHO2.	OR[CO5; DH1] :	$\neg A(x) \lor \underline{R(f_A(x),a)};\ \neg\underline{R(x,y)} \lor S(x,y)\ \vdash$ $\vdash\ \neg A(x) \lor S(f_A(x),a)$	: CO5
AHXO1.	OR[C1; DHXO2] :	$\underline{A(a)};\ \neg\underline{A(x)} \lor \neg S(x,b) \lor T(x,c)\ \vdash\ \neg S(a,c) \lor T(a,c)$	: DHX1
AHXO2.	OR[C2; DHXO1] :	$\underline{R(a,b)};\ \neg\underline{R(x,b)} \lor S(x,c)\ \vdash\ S(a,c)$	: C2
AHXO3.	OR[C2; DHXO3] :	$\underline{R(a,b)};\ \neg A(x) \lor \neg\underline{R(a,b)} \lor S(f_A(x),c)\ \vdash$ $\vdash\ \neg A(x) \lor S(f_A(x),c)$	: CO5
AHXO4.	OR[C3; DHXO2] :	$\underline{A(x)};\ \neg\underline{A(x)} \lor \neg S(x,b) \lor T(x,c)\ \vdash\ \neg S(x,b) \lor T(x,c)$	: DHXO1
AHXO5.	OR[C4; DHXO2] :	$\neg A(x) \lor \underline{B(f_A(x))};\ \neg\underline{B(x)} \lor \neg S(x,b) \lor T(x,c)\ \vdash$ $\vdash\ \neg A(x) \lor \neg S(f_A(x),b) \lor T(f_A(x),c)$	: DHXO4
AHXO6.	OP[CO1; DHX1] :	$\underline{a} \simeq c;\ \neg R(\underline{a},b) \lor S(\underline{a},b)\ \vdash\ \neg R(c,b) \lor S(c,b)$	: DHX1
AHXO7.	OP[CO1; DHX1] :	$\underline{b} \simeq c;\ \neg R(a,\underline{b}) \lor S(a,\underline{b})\ \vdash\ \neg R(a,c) \lor S(a,c)$	: DHX1
AHXO8.	OP[CO1; DHXO1] :	$\underline{a} \simeq c;\ \neg S(x,\underline{a}) \lor T(x,b)\ \vdash\ \neg S(x,c) \lor T(x,b)$	: DHXO1
AHXO9.	OP[CO1; DHXO3] :	$\underline{a} \simeq c;\ \neg A(x) \lor \neg S(\underline{a},b) \lor T(f_A(x),d)\ \vdash$ $\vdash\ \neg A(x) \lor \neg S(c,b) \lor T(f_A(x),d)$	: DHXO3
AHXO10.	OP[CO1; DHXO3] :	$\underline{b} \simeq c;\ \neg A(x) \lor \neg S(a,\underline{b}) \lor T(f_A(x),d)\ \vdash$ $\vdash\ \neg A(x) \lor \neg S(a,c) \lor T(f_A(x),d)$	: DHXO3
AHXO11.	OP[CO1; DHXO4] :	$\underline{a} \simeq c;\ \neg A(x) \lor \neg S(f_A(x),\underline{a}) \lor T(f_A(x),b)\ \vdash$ $\vdash\ \neg A(x) \lor \neg S(f_A(x),c) \lor T(f_A(x),b)$	: DHXO4
AHXO12.	OP[CO2; DHX1] :	$\underline{x} \simeq o;\ \neg R(\underline{a},b) \lor S(\underline{a},b)\ \vdash\ \neg R(o,b) \lor S(o,b)$	: DHX1
AHXO13.	OP[CO2; DHX1] :	$\underline{x} \simeq o;\ \neg R(a,\underline{b}) \lor S(a,\underline{b})\ \vdash\ \neg R(a,o) \lor S(a,o)$	: DHX1
AHXO14.	OP[CO2; DHX3] :	$\underline{x} \simeq o;\ \neg A(x) \lor \neg S(x,\underline{f_A(x)}) \lor T(x,\underline{f_A(x)})\ \vdash$ $\vdash\ \neg A(x) \lor \neg S(x,o) \lor T(x,o)$	: DHXO2

Continued on next page

AHXO15.	OP[CO2; DHXO1] :	$\underline{x} \simeq o;\ \neg S(x, \underline{a}) \vee T(x, b)\ \vdash\ \neg S(x, o) \vee T(x, b)$	: DHXO1
AHXO16.	OP[CO2; DHXO3] :	$\underline{x} \simeq o;\ \neg A(x) \vee \neg S(\underline{a}, b) \vee T(f_A(x), d)\ \vdash$ $\vdash\ \neg A(x) \vee \neg S(o, b) \vee T(f_A(x), d)$	: DHXO3
AHXO17.	OP[CO2; DHXO3] :	$\underline{x} \simeq o;\ \neg A(x) \vee \neg S(a, \underline{b}) \vee T(f_A(x), d)\ \vdash$ $\vdash\ \neg A(x) \vee \neg S(a, o) \vee T(f_A(x), d)$	: DHXO3
AHXO18.	OP[CO2; DHXO4] :	$\underline{x} \simeq o;\ \neg A(x) \vee \neg S(f_A(x), \underline{a}) \vee T(f_A(x), b)\ \vdash$ $\vdash\ \neg A(x) \vee \neg S(f_A(x), o) \vee T(f_A(x), b)$	: DHXO4
AHXO19.	OP[CO2; DHXO4] :	$\underline{x} \simeq o;\ \neg A(x) \vee \neg S(\underline{f_A(x)}, a) \vee T(f_A(x), b)\ \vdash$ $\vdash\ \neg A(x) \vee \neg S(o, a) \vee T(f_A(x), b)$	: DHXO3
AHXO20.	OR[CO3; DHX1] :	$\underline{R(x, b)};\ \neg\underline{R(a, b)} \vee S(a, b)\ \vdash\ S(a, b)$	: C2
AHXO21.	OR[CO3; DHX2] :	$\underline{R(x, b)};\ \neg\underline{R(x, y)} \vee \neg S(x, y) \vee T(x, y)\ \vdash\ \neg S(x, b) \vee T(x, b)$	: DHXO1
AHXO22.	OR[CO3; DHXO1] :	$\underline{R(x, a)};\ \neg\underline{R(x, a)} \vee S(x, b)\ \vdash\ S(x, b)$	: CO3
AHXO23.	OR[CO3; DHXO3] :	$\underline{R(x, b)};\ \neg A(x) \vee \neg\underline{R(a, b)} \vee S(f_A(x), c)\ \vdash$ $\vdash\ \neg A(x) \vee S(f_A(x), c)$	: CO5
AHXO24.	OR[CO3; DHXO4] :	$\underline{R(x, b)};\ \neg A(x) \vee \neg\underline{R(f_A(x), b)} \vee S(f_A(x), c)\ \vdash$ $\vdash\ \neg A(x) \vee S(f_A(x), c)$	: CO5
AHXO25.	OP[CO4; DHX3] :	$\neg A(x) \vee \underline{f_A(x)} \simeq c;\ \neg A(x) \vee \neg S(x, \underline{f_A(x)}) \vee T(x, \underline{f_A(x)})\ \vdash$ $\vdash\ \neg A(x) \vee \neg A(x) \vee \neg S(x, c) \vee T(x, c)\ \vdash \neg A(x) \vee \neg S(x, c) \vee T(x, c)$	: DHXO2
AHXO26.	OP[CO4; DHXO4] :	$\neg A(x) \vee \underline{f_A(x)} \simeq c;\ \neg A(x) \vee \neg R(\underline{f_A(x)}, a) \vee S(f_A(x), b)\ \vdash$ $\vdash\ \neg A(x) \vee \neg A(x) \vee \neg R(c, a) \vee S(f_A(x), b)\ \vdash\ \neg A(x) \vee \neg R(c, a) \vee S(f_A(x), b)$	: DHXO3
AHXO27.	OR[CO5; DHX2] :	$\neg A(x) \vee \underline{R(f_A(x), a)};\ \neg\underline{R(x, y)} \vee \neg S(x, y) \vee T(x, y)\ \vdash$ $\vdash\ \neg A(x) \vee \neg S(f_A(x), a) \vee T(f_A(x), a)$	: DHXO4
AHXO28.	OR[CO5; DHXO1] :	$\neg A(x) \vee \underline{R(f_A(x), a)};\ \neg\underline{R(x, a)} \vee S(x, b)\ \vdash$ $\vdash\ \neg A(x) \vee S(f_A(x), b)$	: CO5
AHXO29.	OR[CO5; DHXO4] :	$\neg A(x) \vee \underline{R(f_A(x), a)};\ \neg A(x) \vee \neg\underline{R(f_A(x), a)} \vee S(f_A(x), b)\ \vdash$ $\vdash\ \neg A(x) \vee \neg A(x) \vee S(f_A(x), b)\ \vdash\ \neg A(x) \vee S(f_A(x), b)$	: CO5
APO1.	OR[C1; DPO1] :	$\underline{B(a)};\ \neg A(x) \vee \neg\underline{B(a)} \vee C(y)\ \vdash\ \neg A(x) \vee C(y)$	: DP6
APO2.	OR[C1; DPO2] :	$\underline{A(b)};\ \neg\underline{A(x)} \vee R(a, y) \vdash\ R(a, x)$	: CP2
APO3.	OR[C1; DPO3] :	$\underline{A(a)};\ \neg\underline{A(x)} \vee R(y, b) \vdash\ R(x, b)$	: CP1
APO4.	OR[C1; DPO4] :	$\underline{B(a)};\ \neg A(x) \vee \neg\underline{B(y)} \vee R(f_A(x), a)\ \vdash\ \neg A(x) \vee R(f_A(x), a)$	: CP3
APO5.	OR[C1; DPO5] :	$\underline{B(a)};\ \neg A(x) \vee \neg\underline{B(y)} \vee R(a, f_A(x))\ \vdash\ \neg A(x) \vee R(a, f_A(x))$	: CP4
APO6.	OR[C1; DPO6] :	$\underline{C(a)};\ \neg A(x) \vee \neg B(y) \vee \neg\underline{C(a)} \vee D(f_A(x))\ \vdash$ $\vdash\ \neg A(x) \vee \neg B(y) \vee D(f_A(x))$	: DP11
APO7.	OR[C3; DPO1] :	$\underline{B(x)};\ \neg A(x) \vee \neg\underline{B(a)} \vee C(y)\ \vdash\ \neg A(x) \vee C(y)$	: CP6
APO8.	OR[C1; DPO2] :	$\underline{A(x)};\ \neg\underline{A(x)} \vee R(a, y) \vdash\ R(a, x)$	: CP2
APO9.	OR[C1; DPO3] :	$\underline{A(x)};\ \neg\underline{A(x)} \vee R(y, b) \vdash\ R(x, b)$	: CP1
APO10.	OR[C3; DPO4] :	$\underline{B(x)};\ \neg A(x) \vee \neg\underline{B(y)} \vee R(f_A(x), a)\ \vdash\ \neg A(x) \vee R(f_A(x), a)$	: CP3
APO11.	OR[C3; DPO5] :	$\underline{B(x)};\ \neg A(x) \vee \neg\underline{B(y)} \vee R(a, f_A(x))\ \vdash\ \neg A(x) \vee R(a, f_A(x))$	: CP4

Continued on next page

APO12.	OR[C3; DPO6] :	$\underline{C(x)};\ \neg A(x) \vee \neg B(y) \vee \neg \underline{C(a)} \vee D(f_A(x))\ \vdash$ $\vdash\ \neg A(x) \vee \neg B(y) \vee D(f_A(x))$	: DP11
APO13.	OR[C4; DPO2] :	$\neg A(x) \vee \underline{B(f_A(x))};\ \neg \underline{B(x)} \vee R(a, y) \vdash\ \neg A(x) \vee R(a, y)$	: DPO2
APO14.	OR[C4; DPO3] :	$\neg A(x) \vee \underline{B(f_A(x))};\ \neg \underline{B(x)} \vee R(y, b) \vdash\ \neg A(x) \vee R(y, b)$	: DPO3
APO15.	OR[C4; DPO4] :	$\neg A(x) \vee \underline{C(f_A(x))};\ \neg B(x) \vee \neg \underline{C(y)} \vee R(g_B(x), a)\ \vdash$ $\vdash\ \neg B(x) \vee \neg A(y) \vee R(g_B(x), a)$	: DPO4
APO16.	OR[C4; DPO5] :	$\neg A(x) \vee \underline{C(f_A(x))};\ \neg A(x) \vee \neg \underline{C(y)} \vee R(a, g_A(x))\ \vdash$ $\vdash\ \neg A(x) \vee \neg C(y) \vee R(a, g_A(x))$	: DPO5
APO17.	OP[CO1; CP2] :	$\underline{a} \simeq b;\ R(\underline{a}, x)\ \vdash\ R(b, x)$	: CP2
APO18.	OP[CO1; CP4] :	$\underline{a} \simeq b;\ \neg A(x) \vee R(\underline{a}, f_A(x))\ \vdash\ \neg A(x) \vee R(b, f_A(x))$	: CP4
APO19.	OP[CO1; DP1] :	$\underline{a} \simeq b;\ \neg A(\underline{a}) \vee B(x)\ \vdash\ \neg A(b) \vee B(x)$	: DP1
APO20.	OP[CO1; DP4] :	$\underline{a} \simeq b;\ \neg A(x) \vee \neg B(\underline{a}) \vee C(f_A(x))\ \vdash$ $\vdash\ \neg A(x) \vee \neg B(b) \vee C(f_A(x))$	: DP4
APO21.	OP[CO1; DPO1] :	$\underline{a} \simeq b;\ \neg A(x) \vee \neg B(\underline{a}) \vee C(y)\ \vdash\ \neg A(x) \vee \neg B(b) \vee C(y)$	: DPO1
APO22.	OP[CO1; DPO6] :	$\underline{a} \simeq b;\ \neg A(x) \vee \neg B(y) \vee \neg C(\underline{a}) \vee D(f_A(x))\ \vdash$ $\vdash\ \neg A(x) \vee \neg B(y) \vee \neg C(b) \vee D(f_A(x))$	: DPO6
APO23.	OP[CO2; CP2] :	$\underline{x} \simeq o;\ R(\underline{a}, x)\ \vdash\ R(o, x)$	: CP2
APO24.	OP[CO2; CP4] :	$\underline{x} \simeq o;\ \neg A(x) \vee R(\underline{a}, f_A(x))\ \vdash\ \neg A(x) \vee R(o, f_A(x))$	: CP4
APO25.	OP[CO2; CP4] :	$\underline{x} \simeq o;\ \neg A(x) \vee R(a, \underline{f_A(x)})\ \vdash\ \neg A(x) \vee R(a, o)$	: DO5
APO26.	OP[CO2; CP6] :	$\underline{x} \simeq o;\ \neg A(x) \vee R(\underline{f_A(x)}, y)\ \vdash\ \neg A(y) \vee R(o, y)$	: DP3
APO27.	OP[CO2; CP7] :	$\underline{x} \simeq o;\ \neg A(y) \vee R(x, \underline{f_A(y)})\ \vdash\ \neg A(x) \vee R(x, o)$	: DO6
APO28.	OP[CO2; CP8] :	$\underline{x} \simeq o;\ \neg A(x) \vee \neg B(y) \vee R(f_A(x), \underline{g_B(y)})\ \vdash$ $\vdash\ \neg A(x) \vee \neg B(y) \vee R(f_A(x), o)$	: DPO4
APO29.	OP[CO2; CP8] :	$\underline{x} \simeq o;\ \neg A(x) \vee \neg B(y) \vee R(\underline{f_A(x)}, g_B(y))\ \vdash$ $\vdash\ \neg B(x) \vee \neg A(y) \vee R(o, g_B(x))$	: DPO5
APO30.	OP[CO2; DP1] :	$\underline{x} \simeq o;\ \neg A(\underline{a}) \vee B(x)\ \vdash\ \neg A(o) \vee B(x)$	: DP1
APO31.	OP[CO2; DP4] :	$\underline{x} \simeq o;\ \neg A(x) \vee \neg B(\underline{a}) \vee C(f_A(x))\ \vdash$ $\vdash\ \neg A(x) \vee \neg B(o) \vee C(f_A(x))$	: DP4
APO32.	OP[CO2; DP5] :	$\underline{x} \simeq o;\ \neg A(x) \vee \neg B(\underline{f_A(x)}) \vee C(a)\ \vdash$ $\vdash\ \neg A(x) \vee \neg B(o) \vee C(a)$	: DO7
APO33.	OP[CO2; DP12] :	$\underline{x} \simeq o;\ \neg A(x) \vee \neg B(\underline{f_A(x)}) \vee C(y)\ \vdash$ $\vdash\ \neg A(x) \vee \neg B(o) \vee C(y)$	: DPO1
APO34.	OP[CO2; DP14] :	$\underline{x} \simeq o;\ \neg A(x) \vee \neg B(y) \vee \neg C(\underline{f_A(x)}) \vee D(g_B(y))\ \vdash$ $\vdash\ \neg B(x) \vee \neg A(y) \vee \neg C(o) \vee D(g_B(x))$	: DPO6
APO35.	OP[CO2; DPO1] :	$\underline{x} \simeq o;\ \neg A(x) \vee \neg B(\underline{a}) \vee C(y)\ \vdash\ \neg A(x) \vee \neg B(o) \vee C(y)$	: DPO1
APO36.	OP[CO2; DPO6] :	$\underline{x} \simeq o;\ \neg A(x) \vee \neg B(y) \vee \neg C(\underline{a}) \vee D(f_A(x))\ \vdash$ $\vdash\ \neg A(x) \vee \neg B(y) \vee \neg C(o) \vee D(f_A(x))$	: DPO6
APO37.	OR[CO3; DP7] :	$\underline{R(x, b)};\ \neg \underline{R(x, y)} \vee A(x)\ \vdash\ A(x)$	: C3
APO38.	OR[CO3; DP8] :	$\underline{R(x, b)};\ \neg \underline{R(x, y)} \vee B(y)\ \vdash\ B(b)$	: C1

Continued on next page

APO39	OP[CO4; CP4] :	$\neg A(x) \vee \underline{f_A(x)} \simeq b;\ \neg A(x) \vee R(a, \underline{f_A(x)})\ \vdash$ $\vdash\ \neg A(x) \vee \neg A(x) \vee R(a, b)\ \vdash\ \neg A(x) \vee R(a, b)$	: DO5
APO40	OP[CO4; CP6] :	$\neg A(x) \vee \underline{f_A(x)} \simeq a;\ \neg A(x) \vee R(\underline{f_A(x)}, y)\ \vdash$ $\vdash\ \neg A(x) \vee \neg A(x) \vee R(a, y) \vdash\ \neg A(x) \vee R(a, y)$	: DPO2
APO41	OP[CO4; CP7] :	$\neg A(x) \vee \underline{f_A(x)} \simeq b;\ \neg A(y) \vee R(x, \underline{f_A(y)})\ \vdash$ $\vdash\ \neg A(x) \vee \neg A(x) \vee R(y, b) \vdash\ \neg A(x) \vee R(y, b)$	: DPO3
APO42.	OP[CO4; CP8] :	$\neg B(x) \vee \underline{g_B(x)} \simeq b;\ \neg A(x) \vee \neg B(y) \vee R(f_A(x), \underline{g_B(y)})\ \vdash$ $\vdash\ \neg B(y) \vee \neg A(x) \vee \neg B(y) \vee R(f_A(x), b)\ \vdash\ \neg A(x) \vee \neg B(y) \vee R(f_A(x), b)$	: DPO4
APO43.	OP[CO4; CP8] :	$\neg A(x) \vee \underline{f_A(x)} \simeq b;\ \neg A(x) \vee \neg B(y) \vee R(\underline{f_A(x)}, g_B(y))\ \vdash$ $\vdash\ \neg A(x) \vee \neg B(x) \vee \neg A(y) \vee R(b, g_B(x))\ \vdash\ \neg B(x) \vee \neg A(y) \vee R(b, g_B(x))$	: DPO5
APO44.	OP[CO4; DP5] :	$\neg A(x) \vee \underline{f_A(x)} \simeq a;\ \neg A(x) \vee \neg B(\underline{f_A(x)}) \vee C(b)\ \vdash$ $\vdash\ \neg A(x) \vee \neg A(x) \vee \neg B(a) \vee C(b)\ \vdash\ \neg A(x) \vee \neg B(a) \vee C(b)$	: DO7
APO45.	OP[CO4; DP12] :	$\neg A(x) \vee \underline{f_A(x)} \simeq b;\ \neg A(x) \vee \neg B(\underline{f_A(x)}) \vee C(y)\ \vdash$ $\vdash\ \neg A(x) \vee \neg A(x) \vee \neg B(b) \vee C(y)\ \vdash\ \neg A(x) \vee \neg B(b) \vee C(y)$	: DPO1
APO46.	OP[CO4; DP14] :	$\neg A(x) \vee \underline{f_A(x)} \simeq b;$ $\neg A(x) \vee \neg B(y) \vee \neg C(\underline{f_A(x)}) \vee D(g_B(y))\ \vdash$ $\vdash\ \neg A(y) \vee \neg B(x) \vee \neg A(y) \vee \neg C(b) \vee D(g_B(x))\ \vdash$ $\vdash\ \neg B(x) \vee \neg A(y) \vee \neg C(b) \vee D(g_B(x))$	: DPO6

Table A.10: An extension of the datalog program for reasoning with nominals in $\mathcal{EL}$

AO1.	CO1(b, c)	$\leftarrow$ C1(A, a), DO1(A, b, c);
AO2.	C1(A, a)	$\leftarrow$ DO2(A, b); CO1(a, b);
AO3.	C3(B)	$\leftarrow$ C1(A, a), DO3(A, B, a);
AO4.	C1(B, b)	$\leftarrow$ C1(A, a), DO4(A, B, b);
AO5.	C2(R, a, b)	$\leftarrow$ C1(A, a), DO5(A, R, a, b);
AO6.	C2(R, a, b)	$\leftarrow$ C1(A, a), DO6(A, R, b);
AO7.	DO4(A, C, b)	$\leftarrow$ C1(B, a), DO7(A, B, C, a, b);
AO8.	D4(A, C)	$\leftarrow$ C1(B, a), DO8(A, B, C, a);
AO9.	C4(A, C, f_A)	$\leftarrow$ C1(B, a), DO9(A, B, C, f_A, a);
AO10.	CO1(b, c)	$\leftarrow$ C3(A), DO1(A, b, c);
AO11.	CO2	$\leftarrow$ C3(A), DO2(A, o);
AO12.	C3(B)	$\leftarrow$ C3(A), DO3(A, B, a);
AO13.	C1(B, b)	$\leftarrow$ C3(A), DO4(A, B, b);
AO14.	C2(R, a, b)	$\leftarrow$ C3(A), DO5(A, R, a, b);
AO15.	CO3(R, a)	$\leftarrow$ C3(A), DO6(A, R, a);
AO16.	DO4(A, C, b)	$\leftarrow$ C3(B), DO7(A, B, C, a, b);
AO17.	D4(A, C)	$\leftarrow$ C3(B), DO8(A, B, C, a);
AO18.	C4(A, C, f_A)	$\leftarrow$ C3(B), DO9(A, B, C, f_A, a);
AO19.	DO1(A, b, c)	$\leftarrow$ C4(A, B, f_A), DO1(B, b, c);
AO20.	CO4(A, f_A, b)	$\leftarrow$ C4(A, B, f_A), DO2(B, b);

Continued on next page

AO21.	$\mathsf{DO4}(A, C, b)$	$\leftarrow \mathsf{C4}(A, B, f_A),\ \mathsf{DO4}(B, C, b);$
AO22.	$\mathsf{DO5}(A, R, a, b)$	$\leftarrow \mathsf{C4}(A, B, f_A),\ \mathsf{DO5}(B, R, a, b);$
AO23.	$\mathsf{CO5}(A, R, f_A, a)$	$\leftarrow \mathsf{C4}(A, B, f_A),\ \mathsf{DO6}(B, R, a);$
AO24.	$\mathsf{C1}(A, b)$	$\leftarrow \mathsf{CO1}(a, b),\ \mathsf{C1}(A, a);$
AO25.	$\mathsf{C2}(R, b, c)$	$\leftarrow \mathsf{CO1}(a, b),\ \mathsf{C2}(R, a, c);$
AO26.	$\mathsf{C2}(R, c, b)$	$\leftarrow \mathsf{CO1}(a, b),\ \mathsf{C2}(R, c, a);$
AO27.	$\mathsf{D1}(A, b)$	$\leftarrow \mathsf{CO1}(a, b),\ \mathsf{D1}(A, a);$
AO28.	$\mathsf{D2}(R, b, c)$	$\leftarrow \mathsf{CO1}(a, b),\ \mathsf{D2}(R, a, c);$
AO29.	$\mathsf{D2}(R, c, b)$	$\leftarrow \mathsf{CO1}(a, b),\ \mathsf{D2}(R, c, a);$
AO30.	$\mathsf{D3}(A, B, b, c)$	$\leftarrow \mathsf{CO1}(a, b),\ \mathsf{D3}(A, B, a, c);$
AO31.	$\mathsf{CO1}(b, c)$	$\leftarrow \mathsf{CO1}(a, b),\ \mathsf{CO1}(a, c);$
AO32.	$\mathsf{CO1}(c, b)$	$\leftarrow \mathsf{CO1}(a, b),\ \mathsf{CO1}(c, a);$
AO33.	$\mathsf{CO3}(R, b)$	$\leftarrow \mathsf{CO1}(a, b),\ \mathsf{CO3}(R, a);$
AO34.	$\mathsf{CO4}(A, f_A, b)$	$\leftarrow \mathsf{CO1}(a, b),\ \mathsf{CO4}(A, f_A, a);$
AO35.	$\mathsf{CO5}(A, R, f_A, b)$	$\leftarrow \mathsf{CO1}(a, b),\ \mathsf{CO5}(A, R, f_A, a);$
AO36.	$\mathsf{DO3}(A, B, b)$	$\leftarrow \mathsf{CO1}(a, b),\ \mathsf{DO3}(A, B, a);$
AO37.	$\mathsf{DO7}(A, B, C, b, c)$	$\leftarrow \mathsf{CO1}(a, b),\ \mathsf{DO7}(A, B, C, a, c);$
AO38.	$\mathsf{DO8}(A, B, C, b)$	$\leftarrow \mathsf{CO1}(a, b),\ \mathsf{DO8}(A, B, C, a);$
AO39.	$\mathsf{DO9}(A, B, C, f_A, b)$	$\leftarrow \mathsf{CO1}(a, b),\ \mathsf{DO9}(A, B, C, f_A, a);$
AO40.	$\mathsf{C1}(A, o)$	$\leftarrow \mathsf{CO2},\ \mathsf{C1}(A, a);$
AO41.	$\mathsf{C2}(R, o, c)$	$\leftarrow \mathsf{CO2},\ \mathsf{C2}(R, a, c);$
AO42.	$\mathsf{DO4}(A, B, o)$	$\leftarrow \mathsf{CO2},\ \mathsf{C4}(A, B, f_A);$
AO43.	$\mathsf{DO6}(A, R, o)$	$\leftarrow \mathsf{CO2},\ \mathsf{C5}(A, R, f_A);$
AO44.	$\mathsf{D1}(A, o)$	$\leftarrow \mathsf{CO2},\ \mathsf{D1}(A, a);$
AO45.	$\mathsf{D2}(R, o, c)$	$\leftarrow \mathsf{CO2},\ \mathsf{D2}(R, a, c);$
AO46.	$\mathsf{D2}(R, c, o)$	$\leftarrow \mathsf{CO2},\ \mathsf{D2}(R, c, a);$
AO47.	$\mathsf{D3}(A, B, o, c)$	$\leftarrow \mathsf{CO2},\ \mathsf{D3}(A, B, a, c);$
AO48.	$\mathsf{DO8}(A, B, C, o)$	$\leftarrow \mathsf{CO2},\ \mathsf{D7}(A, B, C, f_A);$
AO49.	$\mathsf{DO9}(A, B, C, f_A, o)$	$\leftarrow \mathsf{CO2},\ \mathsf{D8}(A, B, C, f_A);$
AO50.	$\mathsf{CO1}(o, c)$	$\leftarrow \mathsf{CO2},\ \mathsf{CO1}(a, c);$
AO51.	$\mathsf{CO1}(c, o)$	$\leftarrow \mathsf{CO2},\ \mathsf{CO1}(c, a);$
AO52.	$\mathsf{CO3}(R, o)$	$\leftarrow \mathsf{CO2},\ \mathsf{CO3}(R, a);$
AO53.	$\mathsf{DO1}(A, o, c)$	$\leftarrow \mathsf{CO2},\ \mathsf{CO4}(A, f_A, c);$
AO54.	$\mathsf{CO4}(A, f_A, o)$	$\leftarrow \mathsf{CO2},\ \mathsf{CO4}(A, f_A, a);$
AO55.	$\mathsf{DO5}(A, R, o, c)$	$\leftarrow \mathsf{CO2},\ \mathsf{CO5}(A, R, f_A, c);$
AO56.	$\mathsf{CO5}(A, R, f_A, o)$	$\leftarrow \mathsf{CO2},\ \mathsf{CO5}(A, R, f_A, a);$
AO57.	$\mathsf{DO3}(A, B, o)$	$\leftarrow \mathsf{CO2},\ \mathsf{DO3}(A, B, a);$
AO58.	$\mathsf{DO7}(A, B, C, o, c)$	$\leftarrow \mathsf{CO2},\ \mathsf{DO7}(A, B, C, a, c);$
AO59.	$\mathsf{DO8}(A, B, C, o)$	$\leftarrow \mathsf{CO2},\ \mathsf{DO8}(A, B, C, a);$
AO60.	$\mathsf{DO9}(A, B, C, f_A, o)$	$\leftarrow \mathsf{CO2},\ \mathsf{DO9}(A, B, C, f_A, a);$
AO61.	$\square$	$\leftarrow \mathsf{CO3}(R, b),\ \mathsf{D2}(R, a, b);$
AO62.	$\mathsf{DO3}(B, A, b)$	$\leftarrow \mathsf{CO3}(R, b),\ \mathsf{D6}(R, B, A);$
AO63.	$\mathsf{DO4}(A, B, a)$	$\leftarrow \mathsf{CO4}(A, f_A, a),\ \mathsf{C4}(A, B, f_A);$
AO64.	$\mathsf{DO6}(A, R, a)$	$\leftarrow \mathsf{CO4}(A, f_A, a),\ \mathsf{C5}(A, R, f_A);$
AO65.	$\mathsf{DO8}(A, B, C, a)$	$\leftarrow \mathsf{CO4}(A, f_A, a),\ \mathsf{D7}(A, B, C, f_A);$
AO66.	$\mathsf{DO9}(A, B, C, f_A, a)$	$\leftarrow \mathsf{CO4}(A, f_A, a),\ \mathsf{D8}(A, B, C, f_A);$

Continued on next page

AO67.	DO1(A,a,b)	← CO4(A,f_A,a), CO4(A,f_A,b);
AO68.	DO5(A,R,a,b)	← CO4(A,f_A,a), CO5(A,R,f_A,b);
AO69.	DO9(A,B,C,f_A,a)	← CO5(A,R,f_A,a), D6(R,B,C);
AHO1.	CO3(S,a)	← CO3(R,a), DH1(R,S);
AHO2.	CO5(A,S,f_A,a)	← CO5(A,R,f_A,a), DH1(R,S);
AHXO1.	DHX1(S,T,a,c)	← C1(A,a), DHXO2(A,S,T,b,c);
AHXO2.	C2(S,a,c)	← C2(R,a,b), DHXO1(R,S,b,c);
AHXO3.	CO5(A,S,f_A,c)	← C2(R,a,b), DHXO3(A,R,S,f_A,a,b,c);
AHXO4.	DHXO1(S,T,b,c)	← C3(A), DHXO2(A,S,T,b,c);
AHXO5.	DHXO4(A,S,T,f_A,b,c)	← C4(A,B,f_A), DHXO2(B,S,T,b,c);
AHXO6.	DHX1(R,S,c,b)	← CO1(a,c), DHX1(R,S,a,b);
AHXO7.	DHX1(R,S,a,c)	← CO1(b,c), DHX1(R,S,a,b);
AHXO8.	DHXO1(S,T,c,b)	← CO1(a,c), DHXO1(S,T,a,b);
AHXO9.	DHXO3(A,S,T,f_A,c,b,d)	← CO1(a,c), DHXO3(A,S,T,f_A,a,b,d);
AHXO10.	DHXO3(A,S,T,f_A,a,c,d)	← CO1(b,c), DHXO3(A,S,T,f_A,a,b,d);
AHXO11.	DHXO4(A,S,T,f_A,c,b)	← CO1(a,c), DHXO4(A,S,T,f_A,a,b);
AHXO12.	DHX1(R,S,o,b)	← CO2, DHX1(R,S,a,b);
AHXO13.	DHX1(R,S,a,o)	← CO2, DHX1(R,S,a,b);
AHXO14.	DHXO2(A,S,T,o,o)	← CO2, DHX3(A,S,T,f_A);
AHXO15.	DHXO1(S,T,o,b)	← CO2, DHXO1(S,T,a,b);
AHXO16.	DHXO3(A,S,T,f_A,o,b,d)	← CO2, DHXO3(A,S,T,f_A,a,b,d);
AHXO17.	DHXO3(A,S,T,f_A,a,o,d)	← CO2, DHXO3(A,S,T,f_A,a,b,d);
AHXO18.	DHXO4(A,S,T,f_A,o,b)	← CO2, DHXO4(A,S,T,f_A,a,b);
AHXO19.	DHXO3(A,S,T,f_A,o,a,b)	← CO2, DHXO4(A,S,T,f_A,a,b);
AHXO20.	C2(S,a,b)	← CO3(R,b), DHX1(R,S,a,b);
AHXO21.	DHXO1(S,T,b,b)	← CO3(R,b), DHX2(R,S,T);
AHXO22.	CO3(S,b)	← CO3(R,a), DHXO1(R,S,a,b);
AHXO23.	CO5(A,S,f_A,c)	← CO3(R,b), DHXO3(A,R,S,f_A,a,b,c);
AHXO24.	CO5(A,S,f_A,c)	← CO3(R,b), DHXO4(A,R,S,f_A,b,c);
AHXO25.	DHXO2(A,S,T,c,c)	← CO4(A,f_A,c), DHX3(A,S,T,f_A);
AHXO26.	DHXO3(A,R,S,f_A,c,a,b)	← CO4(A,f_A,c), DHXO4(A,R,S,f_A,a,b);
AHXO27.	DHXO4(A,S,T,f_A,a,a)	← CO5(A,R,f_A,a), DHX2(R,S,T);
AHXO28.	CO5(A,S,f_A,b)	← CO5(A,R,f_A,a), DHXO1(R,S,a,b);
AHXO29.	CO5(A,S,f_A,b)	← CO5(A,R,f_A,a), DHXO4(A,R,S,f_A,a,b);
APO1.	DP6(A,C)	← C1(B,a), DPO1(A,B,C,a);
APO2.	CP2(R,a)	← C1(A,b), DPO2(A,R,a);
APO3.	CP1(R,b)	← C1(A,a), DPO3(A,R,b);
APO4.	CP3(A,R,f_A,a)	← C1(B,a), DPO4(A,B,R,f_A,a);
APO5.	CP4(A,R,f_A,a)	← C1(B,a), DPO5(A,B,R,f_A,a);
APO6.	DP11(A,B,D,f_A)	← C1(C,a), DPO6(A,B,C,D,f_A,a);
APO7.	DP6(A,C)	← C3(B), DPO1(A,B,C,a);
APO8.	CP2(R,a)	← C3(A), DPO2(A,R,a);
APO9.	CP1(R,b)	← C3(A), DPO3(A,R,b);
APO10.	CP3(A,R,f_A,a)	← C3(B), DPO4(A,B,R,f_A,a);
APO11.	CP4(A,R,f_A,a)	← C3(B), DPO5(A,B,R,f_A,a);

Continued on next page

APO12.	DP11(A, B, D, f_A)	$\leftarrow$ C3(C), DPO6(A, B, C, D, f_A, a);
APO13.	DPO2(A, R, a)	$\leftarrow$ C4(A, B, f_A), DPO2(B, R, a);
APO14.	DPO3(A, R, b)	$\leftarrow$ C4(A, B, f_A), DPO3(B, R, b);
APO15.	DPO4(B, A, R, g_B, a)	$\leftarrow$ C4(A, C, f_A), DPO4(B, C, R, g_B, a);
APO16.	DPO5(A, C, R, g_B, a)	$\leftarrow$ C4(A, C, f_A), DPO5(A, C, R, g_B, a);
APO17.	CP2(R, b)	$\leftarrow$ CO1(a, b), CP2(R, a);
APO18.	CP4(A, R, f_A, b)	$\leftarrow$ CO1(a, b), CP4(A, R, f_A, a);
APO19.	DP1(A, B, b)	$\leftarrow$ CO1(a, b), DP1(A, B, a);
APO20.	DP4(A, B, C, f_A, b)	$\leftarrow$ CO1(a, b), DP4(A, B, C, f_A, a);
APO21.	DPO1(A, B, C, b)	$\leftarrow$ CO1(a, b), DPO1(A, B, C, a);
APO22.	DPO6(A, B, C, D, f_A, b)	$\leftarrow$ CO1(a, b), DPO6(A, B, C, D, f_A, a);
APO23.	CP2(R, o)	$\leftarrow$ CO2, CP2(R, a);
APO24.	CP4(A, R, f_A, o)	$\leftarrow$ CO2, CP4(A, R, f_A, a);
APO25.	DO5(A, R, a, o)	$\leftarrow$ CO2, CP4(A, R, f_A, a);
APO26.	DP3(A, R, a)	$\leftarrow$ CO2(a), CP6(A, R, f_A);
APO27.	DO6(A, R, o)	$\leftarrow$ CO2, CP7(A, R, f_A);
APO28.	DPO4(A, B, R, f_A, o)	$\leftarrow$ CO2, CP8(A, B, R, f_A, g_B);
APO29.	DPO5(B, A, R, g_B, o)	$\leftarrow$ CO2, CP8(A, B, R, f_A, g_B);
APO30.	DP1(A, B, o)	$\leftarrow$ CO2, DP1(A, B, a);
APO31.	DP4(A, B, C, f_A, o)	$\leftarrow$ CO2, DP4(A, B, C, f_A, a);
APO32.	DO7(A, B, C, o, a)	$\leftarrow$ CO2, DP5(A, B, C, f_A, a);
APO33.	DPO1(A, B, C, o)	$\leftarrow$ CO2, DP12(A, B, C, f_A);
APO34.	DPO6(B, A, C, D, g_B, o)	$\leftarrow$ CO2, DP14(A, B, C, D, f_A, g_B);
APO35.	DPO1(A, B, C, o)	$\leftarrow$ CO2, DPO1(A, B, C, a);
APO36.	DPO6(A, B, C, D, f_A, o)	$\leftarrow$ CO2, DPO6(A, B, C, D, f_A, a);
APO37.	C3(A)	$\leftarrow$ CO3(R, b), DP7(R, A);
APO38.	C1(B, b)	$\leftarrow$ CO3(R, b), DP8(R, B);
APO39.	DO5(A, R, a, b)	$\leftarrow$ CO4(A, f_A, b), CP4(A, R, f_A, a);
APO40.	DPO2(A, R, a)	$\leftarrow$ CO4(A, f_A, a), CP6(A, R, f_A);
APO41.	DPO3(A, R, b)	$\leftarrow$ CO4(A, f_A, b), CP7(A, R, f_A);
APO42.	DPO4(A, B, R, f_A, b)	$\leftarrow$ CO4(B, g_B, b), CP8(A, B, R, f_A, g_B);
APO43.	DPO5(B, A, R, g_B, b)	$\leftarrow$ CO4(A, f_A, b), CP8(A, B, R, f_A, g_B);
APO44.	DO7(A, B, C, a, b)	$\leftarrow$ CO4(A, f_A, a), DP5(A, B, C, f_A, b);
APO45.	DPO1(A, B, C, b)	$\leftarrow$ CO4(A, f_A, b), DP12(A, B, C, f_A);
APO46.	DPO6(B, A, C, D, g_B, b)	$\leftarrow$ CO4(A, f_A, b), DP14(A, B, C, D, f_A, g_B);

A.6 Extensions of DL $\mathcal{EL}$ with Restricted Role-Value Maps

In this appendix we describe a resolution-based decision procedure for the extension of $\mathcal{EL}$ with restricted role-value maps that was sketched in subsection 2.5.2. We demonstrate that all new clauses that are obtained for this extensions can be captured by the clause types given in Table A.11. Intuitively, the new atoms $T^a(x)$

Table A.11 Clause types for extensions of $\mathcal{EL}$ with for restricted role-value **m**aps

CM1	$\underline{\boldsymbol{T}^{\boldsymbol{a}}(\boldsymbol{b})}$;	DM1	$\neg\underline{\boldsymbol{A}(\boldsymbol{a})} \vee \boldsymbol{T}^{\boldsymbol{B}}(\boldsymbol{b})$;
CM2	$\underline{\boldsymbol{T}^{\boldsymbol{A}}(\boldsymbol{a})}$;	DM2	$\neg\underline{\boldsymbol{S}(x,\boldsymbol{a})} \vee \boldsymbol{A}(x)$;
CM3	$\neg\boldsymbol{A}(x) \vee \underline{\boldsymbol{T}^{\boldsymbol{B}}(x)}$;	DM3	$\neg\underline{\boldsymbol{S}(x,\boldsymbol{a})} \vee \boldsymbol{T}^{\boldsymbol{A}}(x)$;
CM4	$\boldsymbol{R}^{\neg}(x,y) \vee \underline{\boldsymbol{R}(x,y)}$;	DM4	$\neg\underline{\boldsymbol{S}(x,\boldsymbol{a})} \vee \boldsymbol{T}^{\boldsymbol{b}}(x)$;
		DM5	$\neg\boldsymbol{S}(\boldsymbol{a},x) \vee \neg\underline{\boldsymbol{T}^{\boldsymbol{b}}(x)}$
		DM6	$\neg\boldsymbol{S}(x,y) \vee \neg\underline{\boldsymbol{T}^{\boldsymbol{a}}(y)} \vee \boldsymbol{A}(x)$
		DM7	$\neg\boldsymbol{S}(x,y) \vee \neg\underline{\boldsymbol{T}^{\boldsymbol{a}}(y)} \vee \boldsymbol{H}^{\boldsymbol{A}}(x)$;
		DM8	$\neg\boldsymbol{S}(x,y) \vee \neg\underline{\boldsymbol{T}^{\boldsymbol{a}}(y)} \vee \boldsymbol{H}^{\boldsymbol{b}}(x)$;
		DM9	$\neg\boldsymbol{S}(x,y) \vee \neg\underline{\boldsymbol{T}^{\boldsymbol{A}}(y)} \vee \boldsymbol{B}(x)$;
		DM10	$\neg\boldsymbol{S}(x,y) \vee \neg\underline{\boldsymbol{T}^{\boldsymbol{A}}(y)} \vee \boldsymbol{H}^{\boldsymbol{B}}(x)$;
		DM11	$\neg\underline{\boldsymbol{S}(x,y)} \vee \neg\boldsymbol{A}(y) \vee \boldsymbol{T}^{\boldsymbol{B}}(x)$;
		DM12	$\neg\boldsymbol{S}(x,y) \vee \neg\boldsymbol{T}(y,z) \vee \neg\underline{\boldsymbol{H}^{\neg}(x,z)}$;
		DM13	$\neg\boldsymbol{A}(x) \vee \neg\underline{\boldsymbol{B}(\boldsymbol{f}_{\boldsymbol{A}}(x))} \vee \boldsymbol{T}^{\boldsymbol{C}}(x)$;

have a similar meaning as $T^A(x)$ discussed subsection 2.5.2, and originate from Splitting through New Predicate Symbol, where a are individuals. When atom $T^A(x)$, as has been pointed out, corresponds to concept $\exists T.A$, atom $T^a(x)$ has a semantical meaning $\exists T.\{a\}$.

In Table A.12 we have listed all possible resolution inferences between clauses for this description logic from Table A.11. All inferences are straightforward except for those that employ splitting rule (2.17), which we comment on below.

In inferences AM13 – AM16, TM5 and TM6 we have simulated resolution with the conclusions of compositional axioms, as has been demonstrated for the transitivity axiom in subsection 2.5.2. For this, we have used the "renamed" clauses CM4 and DM12 corresponding to these compositional axioms. Every such inference is simulated by a sequence of two resolution inferences with these clauses, and results in a clause which is split into two clauses using rule (2.17).

The analysis of possible inferences shows that the described set of clauses is closed under ordered resolution, which implies that reasoning tasks for the considered description logic can be solved in polynomial time. The exact complexity bound can be extracted by analysing an extension of datalog program for restricted RVMs given in Table A.13. Applying usual calculations of prefix firings (the number of dots-underlined atoms is bounded by $O(n^4)$), we observe that the complexity of $\mathcal{EL}$ with restricted RVMs is considerably higher than of $\mathcal{EL}$: $O(n^5)$ versus $O(n^3)$.

We haven't investigated further extensions of $\mathcal{EL}$ with restricted RVMs, simple role hierarchies and nominals, but we think that these extensions are fairly straightforward.

Table A.12: Summary of inferences for restricted role-value maps in $\mathcal{EL}$

AM1.	OR[C1; DM1] :	$\underline{A(a)};\ \neg\underline{A(a)} \vee T^B(b) \ \vdash\ T^B(b)$	: CM2
AM2.	OR[C2; DM2] :	$\underline{R(a,b)};\ \neg\underline{R(x,b)} \vee A(x) \ \vdash\ A(a)$	: C1
AM3.	OR[C2; DM3] :	$\underline{R(a,b)};\ \neg\underline{R(x,b)} \vee T^A(x) \ \vdash\ T^A(a)$	: CM2
AM4.	OR[C2; DM4] :	$\underline{R(a,b)};\ \neg\underline{R(x,b)} \vee T^c(x) \ \vdash\ T^c(a)$	: CM1
AM5.	OR[C2; DM11] :	$\underline{R(a,b)};\ \neg\underline{R(x,y)} \vee \neg A(y) \vee T^B(x) \ \vdash\ \neg A(b) \vee T^B(a)$	: DM1
AM6.	OR[C3; DM1] :	$\underline{A(x)};\ \neg\underline{A(a)} \vee T^B(b) \ \vdash\ T^B(b)$	: CM2
AM7.	OR[CM1; DM5] :	$\underline{T^a(b)};\ \neg S(c,x) \vee \neg\underline{T^a(x)} \vdash\ \neg S(c,b)$	: D2
AM8.	OR[CM1; DM6] :	$\underline{T^a(b)};\ \neg S(x,y) \vee \neg\underline{T^a(y)} \vee A(x) \vdash\ \neg S(x,b) \vee A(x)$	: DM2
AM9.	OR[CM1; DM7] :	$\underline{T^a(b)};\ \neg S(x,y) \vee \neg\underline{T^a(y)} \vee H^A(x) \ \vdash\ \neg S(x,b) \vee H^A(x)$	: DM3
AM10.	OR[CM1; DM8] :	$\underline{T^a(b)};\ \neg S(x,y) \vee \neg\underline{T^a(y)} \vee H^c(x) \ \vdash\ \neg S(x,b) \vee H^c(x)$	: DM4
AM11.	OR[CM2; DM9] :	$\underline{T^A(a)};\ \neg S(x,y) \vee \neg\underline{T^A(y)} \vee B(x) \ \vdash\ \neg S(x,a) \vee B(x)$	: DM2
AM12.	OR[CM2; DM10] :	$\underline{T^A(a)};\ \neg S(x,y) \vee \neg\underline{T^A(y)} \vee H^B(x) \ \vdash\ \neg S(x,a) \vee H^B(x)$	: DM3
AM13.	OR[CM4; D2] :	$H^{\neg}(x,y) \vee \underline{H(x,y)};\ \neg\underline{H(a,b)} \ \vdash\ H^{\neg}(a,b);$	
	OR[· ; DM12] :	$\underline{H^{\neg}(a,b)};\ \neg S(x,y) \vee \neg T(y,z) \vee \neg\underline{H^{\neg}(x,z)} \ \vdash$	
		$\vdash\ \llbracket \neg S(a,x) \vee \neg T(x,b) \rrbracket \ \vdash\ \neg S(a,x) \vee \neg T^b(x)$	: DM5
		$\vdash\ \neg T(x,b) \vee T^b(x)$	: DM4
AM14.	OR[CM4; DM2] :	$H^{\neg}(x,y) \vee \underline{H(x,y)};\ \neg\underline{H(x,a)} \vee A(x) \ \vdash\ H^{\neg}(x,a) \vee A(x);$	
	OR[· ; DM12] :	$\underline{H^{\neg}(x,a)} \vee A(x);\ \neg S(x,y) \vee \neg T(y,z) \vee \neg\underline{H^{\neg}(x,z)} \ \vdash$	
		$\vdash\ \llbracket \neg S(x,y) \vee \neg T(y,a) \vee A(x) \rrbracket \ \vdash\ \neg S(x,y) \vee \neg T^a(y) \vee A(x)$	: DM6
		$\vdash\ \neg T(x,a) \vee T^a(x)$	: DM4
AM15.	OR[CM4; DM3] :	$H^{\neg}(x,y) \vee \underline{H(x,y)};\ \neg\underline{H(x,a)} \vee R^A(x) \ \vdash\ H^{\neg}(x,a) \vee R^A(x);$	
	OR[· ; DM12] :	$\underline{H^{\neg}(x,a)} \vee R^A(x);\ \neg S(x,y) \vee \neg T(y,z) \vee \neg\underline{H^{\neg}(x,z)} \ \vdash$	
		$\vdash\ \llbracket \neg S(x,y) \vee \neg T(y,a) \vee R^A(x) \rrbracket \ \vdash\ \neg S(x,y) \vee \neg T^a(y) \vee R^A(x)$	: DM7
		$\vdash\ \neg T(x,a) \vee T^a(x)$	: DM4
AM16.	OR[CM4; DM4] :	$H^{\neg}(x,y) \vee \underline{H(x,y)};\ \neg\underline{H(x,a)} \vee R^b(x) \ \vdash\ H^{\neg}(x,a) \vee R^b(x);$	
	OR[· ; DM12] :	$\underline{H^{\neg}(x,a)} \vee R^b(x);\ \neg S(x,y) \vee \neg T(y,z) \vee \neg\underline{H^{\neg}(x,z)} \ \vdash$	
		$\vdash\ \llbracket \neg S(x,y) \vee \neg T(y,a) \vee R^b(x) \rrbracket \ \vdash\ \neg S(x,y) \vee \neg T^a(y) \vee R^b(x)$	: DM8
		$\vdash\ \neg T(x,a) \vee T^a(x)$	: DM4
TM1.	OR[C3; DM13] :	$\underline{B(x)};\ \neg A(x) \vee \neg\underline{B(f_A(x))} \vee T^C(x) \ \vdash\ \neg A(x) \vee T^C(x)$	: CM3
TM2.	OR[C4; DM13] :	$\neg A(x) \vee \underline{B(f_A(x))};\ \neg A(x) \vee \neg\underline{B(f_A(x))} \vee T^C(x) \ \vdash$	
		$\vdash\ \neg A(x) \vee \neg A(x) \vee T^C(x) \ \vdash\ \neg A(x) \vee T^C(x)$	: CM3
TM3.	OR[C5; DM11] :	$\neg A(x) \vee \underline{R(x,f_A(x))};\ \neg\underline{R(x,y)} \vee \neg B(y) \vee T^C(x) \ \vdash$	
		$\vdash\ \neg A(x) \vee \neg B(f_A(x)) \vee T^C(x)$	: DM13
TM5.	OR[CM3; DM9] :	$\neg A(x) \vee \underline{T^B(x)};\ \neg S(x,y) \vee \neg\underline{T^B(y)} \vee C(x) \ \vdash$	
		$\vdash\ \neg S(x,y) \vee \neg A(y) \vee C(x)$	: D6

Continued on next page

$\mathsf{TM4.}$ $\mathtt{OR}[\mathsf{CM3};\mathsf{DM10}] : \neg A(x) \vee \underline{T^B(x)};\ \neg S(x,y) \vee \neg \underline{T^B(y)} \vee H^C(x)\ \vdash$
$\vdash\ \neg S(x,y) \vee \neg A(y) \vee H^C(x)\ : \mathsf{DM11}$

$\mathsf{TM5.}$ $\mathtt{OR}[\mathsf{CM4};\mathsf{D6}] : \quad H^\neg(x,y) \vee \underline{H(x,y)};\ \neg \underline{H(x,y)} \vee \neg B(y) \vee A(x)\ \vdash$
$\vdash\ H^\neg(x,y) \vee \neg B(y) \vee A(x);$
$\mathtt{OR}[\ \cdot\ ;\mathsf{DM12}] : \quad \underline{H^\neg(x,y)} \vee \neg A(y) \vee B(x);\ \neg S(x,y) \vee \neg T(y,z) \vee \neg \underline{H^\neg(x,z)}\ \vdash$
$\vdash\ \llbracket \neg S(x,y) \vee \neg T(y,z) \vee \neg A(y) \vee B(x) \rrbracket\ \vdash\ \neg S(x,y) \vee \neg T^A(y) \vee B(x)\ : \mathsf{DM9}$
$\vdash\ \neg T(x,y) \vee \neg A(y) \vee T^A(x)\ : \mathsf{DM11}$

$\mathsf{TM6.}$ $\mathtt{OR}[\mathsf{CM4};\mathsf{DM11}] : H^\neg(x,y) \vee \underline{H(x,y)};\ \neg \underline{H(x,y)} \vee \neg A(y) \vee R^B(x)\ \vdash$
$\vdash\ H^\neg(x,y) \vee \neg A(y) \vee R^B(x);$
$\mathtt{OR}[\ \cdot\ ;\mathsf{DM12}] : \quad \underline{H^\neg(x,y)} \vee \neg A(y) \vee R^B(x);\ \neg S(x,y) \vee \neg T(y,z) \vee \neg \underline{H^\neg(x,z)}\ \vdash$
$\vdash\ \llbracket \neg S(x,y) \vee \neg T(y,z) \vee \neg A(y) \vee R^B(x) \rrbracket\ \vdash\ \neg S(x,y) \vee \neg T^A(y) \vee R^B(x)\ : \mathsf{DM10}$
$\vdash\ \neg T(x,y) \vee \neg A(y) \vee T^A(x)\ : \mathsf{DM11}$

Table A.13: An extension of the datalog program for reasoning with restricted role-value maps in $\mathcal{EL}$

AM1.	$\mathsf{CM2}(B,T,b)$	$\leftarrow \mathsf{C1}(\boldsymbol{A},\boldsymbol{a}),\ \mathsf{DM1}(\boldsymbol{A},B,T,\boldsymbol{a},b);$
AM2.	$\mathsf{C1}(A,a)$	$\leftarrow \mathsf{C2}(\boldsymbol{R},\underline{a,\boldsymbol{b}}),\ \mathsf{DM2}(\boldsymbol{R},\boldsymbol{b},A);$
AM3.	$\mathsf{CM2}(A,T,a)$	$\leftarrow \mathsf{C2}(\boldsymbol{R},\underline{a,\boldsymbol{b}}),\ \mathsf{DM3}(\boldsymbol{R},\boldsymbol{b},A,T);$
AM4.	$\mathsf{CM1}(c,T,a)$	$\leftarrow \mathsf{C2}(\boldsymbol{R},\underline{a,b}),\ \mathsf{DM4}(\boldsymbol{R},b,c,T);$
AM5.	$\mathsf{DM1}(A,B,T,b,a)$	$\leftarrow \mathsf{C2}(\boldsymbol{R},\underline{a,b}),\ \mathsf{DM11}(\boldsymbol{R},A,B,T);$
AM6.	$\mathsf{CM2}(B,T,b)$	$\leftarrow \mathsf{C3}(\boldsymbol{A}),\ \mathsf{DM1}(\boldsymbol{A},B,T,a,b);$
AM7.	$\mathsf{D2}(S,c,b)$	$\leftarrow \mathsf{CM1}(\boldsymbol{a},\boldsymbol{T},b),\ \mathsf{DM5}(S,c,\boldsymbol{a},\boldsymbol{T});$
AM8.	$\mathsf{DM2}(S,b,A)$	$\leftarrow \mathsf{CM1}(\boldsymbol{a},\boldsymbol{T},b),\ \mathsf{DM6}(S,\boldsymbol{a},\boldsymbol{T},A);$
AM9.	$\mathsf{DM3}(S,b,A,H)$	$\leftarrow \mathsf{CM1}(\boldsymbol{a},\boldsymbol{T},b),\ \dotuline{\mathsf{DM7}(S,\boldsymbol{a},\boldsymbol{T},A,H)};$
AM10.	$\mathsf{DM4}(S,b,c,H)$	$\leftarrow \mathsf{CM1}(\boldsymbol{a},\boldsymbol{T},b),\ \dotuline{\mathsf{DM8}(S,\boldsymbol{a},\boldsymbol{T},c,H)};$
AM11.	$\mathsf{DM2}(S,a,B)$	$\leftarrow \mathsf{CM2}(\boldsymbol{A},\boldsymbol{T},a),\ \mathsf{DM9}(S,\boldsymbol{A},\boldsymbol{T},B);$
AM12.	$\mathsf{DM3}(S,a,B,H)$	$\leftarrow \mathsf{CM2}(\boldsymbol{A},\boldsymbol{T},a),\ \dotuline{\mathsf{DM10}(S,\boldsymbol{A},\boldsymbol{T},B,H)};$
AM13.	$\mathsf{DM5}(S,a,b,T)$	$\leftarrow \mathsf{D2}(\boldsymbol{H},a,b),\ \underline{\mathsf{DM12}(S,T,\boldsymbol{H})};$
AM13′.	$\mathsf{DM4}(T,b,b,T)$	$\leftarrow \mathsf{D2}(\boldsymbol{H},a,b),\ \underline{\mathsf{DM12}(S,T,\boldsymbol{H})};$
AM14.	$\mathsf{DM6}(S,a,T,A)$	$\leftarrow \mathsf{DM2}(\boldsymbol{H},a,A),\ \underline{\mathsf{DM12}(S,T,\boldsymbol{H})};$
AM14′.	$\mathsf{DM4}(T,a,a,T)$	$\leftarrow \mathsf{DM2}(\boldsymbol{H},a,A),\ \underline{\mathsf{DM12}(S,T,\boldsymbol{H})};$
AM15.	$\mathsf{DM7}(S,a,T,A,R)$	$\leftarrow \mathsf{DM3}(\boldsymbol{H},a,A,R),\ \underline{\mathsf{DM12}(S,T,\boldsymbol{H})};$
AM15′.	$\mathsf{DM4}(T,a,a,T)$	$\leftarrow \mathsf{DM3}(\boldsymbol{H},a,A,R),\ \underline{\mathsf{DM12}(S,T,\boldsymbol{H})};$
AM16.	$\mathsf{DM8}(S,a,T,b,R)$	$\leftarrow \mathsf{DM4}(\boldsymbol{H},a,b,R),\ \underline{\mathsf{DM12}(S,T,\boldsymbol{H})};$
AM16′.	$\mathsf{DM4}(T,a,a,T)$	$\leftarrow \mathsf{DM4}(\boldsymbol{H},a,b,R),\ \underline{\mathsf{DM12}(S,T,\boldsymbol{H})};$
TM1.	$\mathsf{CM3}(A,C,T)$	$\leftarrow \mathsf{C3}(\boldsymbol{B}),\ \mathsf{DM13}(A,\boldsymbol{B},C,T,f_A);$
TM2.	$\mathsf{CM3}(A,C,T)$	$\leftarrow \mathsf{C4}(\boldsymbol{A},\boldsymbol{B},\boldsymbol{f_A}),\ \mathsf{DM13}(\boldsymbol{A},\boldsymbol{B},C,T,\boldsymbol{f_A});$
TM3.	$\mathsf{DM13}(A,B,C,T,f_A)$	$\leftarrow \mathsf{C5}(A,\boldsymbol{R},f_A),\ \mathsf{DM11}(\boldsymbol{R},B,C,T);$
TM5.	$\mathsf{D6}(S,A,C)$	$\leftarrow \mathsf{CM3}(A,\boldsymbol{B},\boldsymbol{T}),\ \mathsf{DM9}(S,\boldsymbol{B},\boldsymbol{T},C);$
TM4.	$\mathsf{DM11}(S,A,C,H)$	$\leftarrow \mathsf{CM3}(A,\boldsymbol{B},\boldsymbol{T}),\ \dotuline{\mathsf{DM10}(S,\boldsymbol{B},\boldsymbol{T},C,H)};$

Continued on next page

TM5.	$\mathsf{DM9}(S, A, T, B)$	$\leftarrow \mathsf{D6}(\boldsymbol{H}, B, A),\ \underline{\mathsf{DM12}(S, T, \boldsymbol{H})}$;
TM5′.	$\mathsf{DM11}(T, A, A, T)$	$\leftarrow \mathsf{D6}(\boldsymbol{H}, B, A),\ \underline{\mathsf{DM12}(S, T, \boldsymbol{H})}$;
TM6.	$\mathsf{DM10}(S, A, T, B, R)$	$\leftarrow \mathsf{DM11}(\boldsymbol{H}, A, B, R),\ \underline{\mathsf{DM12}(\boldsymbol{S}, T, \boldsymbol{H})}$;
TM6′.	$\mathsf{DM11}(T, A, A, T)$	$\leftarrow \mathsf{DM11}(\boldsymbol{H}, A, B, R),\ \underline{\mathsf{DM12}(\boldsymbol{S}, T, \boldsymbol{H})}$;

A.7 Prolog Programs for Reasoning in DL $\mathcal{EL}$

In this appendix we include the source codes of two prolog programs that were used in experiments described in section 2.6. The first program implements the completion algorithm given in (2.4). The second is based on the datalog program given in Table 2.7. Both programs process a TBox given by a list of concept definitions $C_1 \doteq C_2$ according to the syntax:

`define_concept(`C_1`,`C_2`).`

where C_1 and C_2 are concepts that are defined by the grammar:

CN ::= a | `and(`C_1`,`C_2`)` | `some(`r`,`C_1`)` .

where $C_1, C_2 \in CN$, a is any concept name identifier, and r is any role name identifier (both are strings starting with a small letter). Both programs process the input definitions by first introducing simple definitions for every compound subconcept. After that, the respective completion rules are applied.

In lines 4 – 5 of the programs there are some compiler options for XSB prolog system specified, which turn on automatic optimisations and static analysis for determining tabled predicates (line 4), and additional declarations in order to cache the set of subconcepts (line 5).

It is well-known that the order of atoms in the body of a prolog rules may have a dramatical impact on the running times of logical programs. For some rules we have manually rearranged the atoms in their bodies to obtain a better performance.

Classification of a TBox is done by issuing command “`classify.`” This forces XSB to compute *all* pairs of concepts (A, B) such that “`subsumes(`A`,`B`).`” holds. After that, command “`print.`” prints all computed subsumption relations.

XSB+CR

```
1  %% A prolog program for classification of EL-terminologies
2  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3  % Options for enabling tabling in XSB prolog
4  :- compiler_options([optimize,auto_table]).
5  :- table(sub_c/1). % optimization of simplification
6
7  /* Simplification */
8
9  sub_c(X,X).  % sub concept
```

```
sub_c(C,and(X,Y)) :- sub_c(C,X).
sub_c(C,and(X,Y)) :- sub_c(C,Y).
sub_c(C,some(R,X)) :- sub_c(C,X).
sub_c(C) :- define_concept(X,Y), sub_c(C,X).
sub_c(C) :- define_concept(Y,X), sub_c(C,X).

/* Definitions of Atoms */

in(A,B) :- define_concept(A,B).
in(A,B) :- define_concept(B,A).

in(and(A,B),A) :- sub_c(and(A,B)).
in(and(A,B),B) :- sub_c(and(A,B)).
and_in(A,B,and(A,B)) :- sub_c(and(A,B)).

some_in(R,B,some(R,B)) :- sub_c(some(R,B)).
in_some(some(R,B),R,B) :- sub_c(some(R,B)).

/* Datalog Program */

subsumes(C,C) :- sub_c(C).
subsumes(top,C) :- sub_c(C).

subsumes(D,C) :- in(Ca,D), subsumes(Ca,C).
subsumes(D,C) :- and_in(Ca,Cb,D), subsumes(Ca,C), subsumes(Cb,C).
reachable_R(C,D,R) :- in_some(Ca,R,D), subsumes(Ca,C).
subsumes(E,C) :- some_in(R,Da,E), subsumes(Da,D), reachable_R(C,D,R).

/* Computing a subsumption relation */

classify :- subsumes(A,B),fail.  % compute subsumption

% printing subsumption for non-equal concept names

print_subsumes(A,B) :- subsumes(A,B),A\==B,atom(A),atom(B),
                        writeq(subsumes(A,B)),nl,fail.

print :-  print_subsumes(A,B).
```

XSB+OR

```
%% A prolog program for classification of EL-terminologies
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Options for enabling tabling in XSB prolog
:- compiler_options([optimize,auto_table]).
:- table(sub_c/1). % optimization of simplification

/* Simplification */

```

```
sub_c(X,X).  % sub concept
sub_c(C,and(X,Y)) :- sub_c(C,X).
sub_c(C,and(X,Y)) :- sub_c(C,Y).
sub_c(C,some(R,X)) :- sub_c(C,X).
sub_c(C) :- define_concept(X,Y), sub_c(C,X).
sub_c(C) :- define_concept(Y,X), sub_c(C,X).

/* CNF-translation */

d4(A,B) :- define_concept(A,B).
d4(A,B) :- define_concept(B,A).

d4(and(A,B),A) :- sub_c(and(A,B)).
d4(and(A,B),B) :- sub_c(and(A,B)).
d5(A,B,and(A,B)) :- sub_c(and(A,B)).

c5(some(R,B),R,f(R,B)) :- sub_c(some(R,B)).
c4(some(R,B),B,f(R,B)) :- sub_c(some(R,B)).
d6(R,B,some(R,B)) :- sub_c(some(R,B)).

c1(A,Xa) :- instance(Xa,A).
c2(R,Xa,Xb) :- relation(Xa,Xb,R).

/* Datalog Program */

bot :- c1(A,Xa), d1(A,Xa).
c1(B,Xb) :- c1(A,Xa), d3(A,B,Xa,Xb).
c1(B,Xa) :- c1(A,Xa), d4(A,B).
d3(B,C,Xa,Xa) :- c1(A,Xa), d5(A,B,C).
bot :- c2(R,Xa,Xb), d2(R,Xa,Xb).
d3(B,A,Xb,Xa) :- c2(R,Xa,Xb), d6(R,B,A).
bot :- c3(A), d1(A,Xa).
c1(B,Xb) :- c3(A), d3(A,B,Xa,Xb).

c3(B) :- c3(A), d4(A,B).
d4(B,C) :- c3(A), d5(A,B,C).
d4(B,C) :- c3(A), d7(B,A,C,F).
c4(B,C,F) :- c3(A), d8(B,A,C,F).
c4(A,C,F) :- d4(B,C), c4(A,B,F).
d8(A,C,D,F) :-  d5(B,C,D), c4(A,B,F).
d4(A,C) :-  c4(A,B,F), d7(A,B,C,F).
c4(A,C,F) :-  d8(A,B,C,F), c4(A,B,F).
d7(A,B,C,F) :- d6(R,B,C), c5(A,R,F).

/* Computing a subsumption relation */

sb(A,A) :- sub_c(A).
subsumes(B,A) :- c3(B).
subsumes(B,A) :- sb(B,A).

sb(E,A) :- d4(C,E), sb(C,A).
sb(E,A) :- d5(D,C,E), sb(C,A), sb(D,A).
sb(E,A) :- d5(D,C,E), c3(C), sb(D,A).

classify :- subsumes(A,B),fail.  % compute subsumption

% printing subsumption for non-equal concept names

print_subsumes(A,B) :-         subsumes(A,B),A\==B,atom(A),atom(B),
                        writeq(subsumes(A,B)),nl,fail.
print :-  print_subsumes(A,B).

/* Computing queries */

instances(Xa,A) :- c1(A,Xa),atom(A).
relations(Xa,R,Xb) :- c2(R,Xa,Xb).

print_instances(Xa,A) :- instances(Xa,A),
                         writeq(instance(Xa,A)),nl,fail.

print_relations(Xa,R,Xb) :- relations(Xa,R,Xb),
                            writeq(relation(Xa,R,Xb)),nl,fail.
```

Below we give a simple session demonstrating how to use the last program for querying of a knowledge base. For this we consider a simple terminology of human relations described in Table 2.1. This terminology is represented in our input syntax as follows:

terminology ‘‘Humans’’

```
:- [el].

/* T-Box */

define_concept(man, and(human,male) ).
define_concept(parent, and(human, some(has_child,human) ) ).
define_concept(father, and(man, some(has_child,human) ) ).
define_concept(grandfather, and(man, some(has_child,parent) ) ).

/* A-Box */

instance(john,man).
instance(bill,father).
relation(john,bill,has_child).
```

When we load this terminology into XSB, this automatically loads a reasoner for $\mathcal{EL}$, which is in our case XSB+OR given above. The queries discussed in section 2.1 are evaluated as follows:

```
XSB Version 2.7 (Kinryo) of January 4, 2005
[sparc-sun-solaris2.9; mode: optimal; engine: slg-wam; gc: indirection;
                                                          scheduling: local]
| ?- [humans].
[humans loaded]
[el loaded]

yes
| ?- classify.

no
| ?- subsumes(parent,father).

yes
| ?- print.
subsumes(man,grandfather)
subsumes(man,father)
subsumes(parent,grandfather)
subsumes(parent,father)
subsumes(father,grandfather)
subsumes(male,man)
subsumes(male,grandfather)
subsumes(male,father)
subsumes(human,man)
subsumes(human,grandfather)
subsumes(human,parent)
subsumes(human,father)

no
| ?- instances(john,grandfather).

yes
| ?- print_instances(X,human).
instance(bill,human)
instance(john,human)

no
| ?-
```

Appendix B

Schemes of Expressions and Clauses

To describe clause classes and inferences between them in saturation-based decision procedures, we introduce a special language. Using this language, sets of expressions and clauses can be described in a concise form by means of *scheme expressions*, or shortly *schemes*. The goal of the scheme notation is to visualize case analysis of possible inferences between clauses in proving termination for different saturation strategies.

Formulas from decidable fragments usually correspond to clauses having certain structural properties, characterized by functional depth, occurrences of variables, functional and predicate symbols in clauses. This structural information plays a crucial role for showing decidability of these fragments by ordered resolution or other calculi. So far, there was no notation for this sort of information and traditionally, the natural language has been used for describing saturation-based decision procedures [Joyner Jr., 1976; Fermüller et al., 1993; Bachmair et al., 1993*b*; Ganzinger & de Nivelle, 1999; Hustadt & Schmidt, 1999; Ganzinger et al., 2001; de Nivelle & Pratt-Hartmann, 2001; de Nivelle & de Rijke, 2003]. This worked well for relatively *simple* clause classes. In this theses, we extend many decidability results in a non-trivial way and obtain decision procedures with large number of nuances. Thus, compact and readable notation is required to give a detailed description of these decision procedures.

We came up with a scheme notation that is on one hand, subsumes the standard notation for expressions and clauses, and on the other hand, gives a possibility of defining classes of clauses using additional constructors. The scheme notation has been designed mostly for presentation purposes (to write down clause classes and case analyses compactly in the proofs) and is not completely formal in the moment. Nonetheless, it allows to give a comprehensive account for decision procedures described in the thesis.

In the next sections we introduce the scheme notation and give a semantic for it. Many examples demonstrating the usage and meaning of scheme expressions are provided as well.

B.1 Signature Parameters and the Choice Operator

To represent clause classes independent of a signature, we use special elements called *signature parameters*. Intuitively a signature parameter is a meta-variable ranging over a set of signature elements (predicate or functional symbols). For example, given a signature of arithmetics $\Sigma_{\mathcal{A}} = (\{\mathtt{Nat}^1\}, \{+^2, \times^2, \mathtt{s}^1, \mathtt{exp}^1, \mathtt{zero}^0\})$ consisting of the unary predicate symbol $\mathtt{Nat}$ and the functional symbols: binary $+$ and $\times$ representing the arithmetical operations, unary $\mathtt{s}$ and $\mathtt{exp}$ representing the successor and exponential functions, and a constant $\mathtt{zero}$, we can introduce a signature parameter $f ::= +|\times|\mathtt{s}|\mathtt{exp}|\mathtt{zero}$, that ranges over all functional symbols (and constants). Now we can use the expression $f(x,y)$ for representing the terms $+(x,y)$ or $\times(x,y)$. Note that $f(x,y)$ does not represent $\mathtt{s}(x,y)$, $\mathtt{exp}(x,y)$ or $\mathtt{zero}(x,y)$ because they are not well-formed terms. Additionally, one can introduce parameters for the predicate symbols $a ::= \mathtt{Nat}$; for the binary and unary functional symbols: $f_2 ::= +|\times$; $f_1 ::= \mathtt{s}|\mathtt{exp}$; and for the constants $c ::= \mathtt{zero}$. These, parameters can be used, say, to form the expression $f_2(f_1(x), f_1(y))$ that represents any of the terms $+(\mathtt{s}(x), \mathtt{s}(y))$, $\times(\mathtt{s}(x), \mathtt{s}(y))$, $+(\mathtt{exp}(x), \mathtt{exp}(y))$ and $\times(\mathtt{exp}(x), \mathtt{exp}(y))$, but not, say, the terms $+(\mathtt{s}(x), \mathtt{exp}(y))$ or $\times(\mathtt{exp}(x), \mathtt{s}(y))$, since every occurrence of the parameter f_1 should correspond to the same functional symbol. For the same reason the scheme $f(f(x), f(y))$ does not represent any terms, since no functional symbol can be binary and unary at the same time.

Sometimes it is desirable to evaluate different occurrences of a parameter by different signature elements. One can introduce several variants of parameters for all those occurrence, but this is not very convenient, since the mechanism of renaming should be provided as well. Instead of doing this, we will limit the scope of a parameter, so that the same parameter name can be reused. The *bounding operator* $\langle \ldots \rangle$ limits the scope of every parameter within its region. The parameters whose scope is not bounded in a scheme are called *free parameters*. For example, the term scheme $f\langle(f(x), f(y))\rangle$ represents the same terms as the scheme $f_2(f_1(x), f_1(y))$ considered above. We mostly will make use of a special case of the bounding operator, when a parameter is bounded immediately: $\langle f \rangle$. Such expressions will be called *bounded parameters* and we shorten them to $\hat{f}$. For example, the term $\hat{f}(\hat{f}(x), \hat{f}(y))$ represents any of the terms $+(\mathtt{s}(x), \mathtt{exp}(y))$, $+(\mathtt{s}(x), \mathtt{s}(y))$, $+(\mathtt{exp}(x), \mathtt{s}(y))$, $+(\mathtt{exp}(x), \mathtt{exp}(y))$, $\times(\mathtt{s}(x), \mathtt{exp}(y))$, ...etc.

Using signature parameters one can represent sets of terms having similar structure. We augment the scheme language with the additional constructor that allows

Table B.1 Summary for the usage of signature parameters in schemes

$\Sigma_{\mathcal{A}} = (\{\mathtt{Nat}^1\}, \{+^2, \times^2, \mathtt{s}^1, \mathtt{exp}^1, \mathtt{zero}^0\})$

$f ::= +|\times|\mathtt{s}|\mathtt{exp}|\mathtt{zero}$; $a ::= \mathtt{Nat}$; $f_2 ::= +|\times$; $f_1 ::= \mathtt{s}|\mathtt{exp}$; $c ::= \mathtt{zero}$.

schemes	**represent**	**do not represent,** ***why***
$f(x,y)$	$+(x,y)$ $\times(x,y)$	$\mathtt{s}(x,y)$: *not well-formed* $\mathtt{zero}(x,y)$ *expressions*
$f(x,x)$	$+(x,x)$	$\mathtt{s}(x)$: *arguments should match exactly*
$f_2(f_1(x), f_1(y))$ $\equiv f\langle(f(x), f(y))\rangle$	$+(\mathtt{s}(x), \mathtt{s}(y))$ $\times(\mathtt{exp}(x), \mathtt{exp}(y))$	$+(\mathtt{s}(x), \mathtt{exp}(y))$: f_1 *is a free* $\times(\mathtt{exp}(x), \mathtt{s}(y))$ *parameter*
$\hat{f}(\hat{f}(x), \hat{f}(y))$	$+(\mathtt{s}(x), \mathtt{exp}(y))$ $\times(\mathtt{s}(x), \mathtt{s}(y))$	

one to represent sets of terms having different structures.

Given term schemes $\boldsymbol{\hat{t}_1}$ and $\boldsymbol{\hat{t}_2}$ we can construct a new term scheme $(\boldsymbol{\hat{t}_1}|\boldsymbol{\hat{t}_2})$ using the *choice operator* $(\cdot|\cdot)$. This term scheme represents every clause that is represented by either t_1 or t_2. For example, the clause scheme $(f(x)|f(x,y))$ represents exactly the terms $\{\mathtt{exp}(x), \mathtt{s}(x), +(x,y), \times(x,y)\}$ over signature $\Sigma_{\mathcal{A}}$. The choice operator does not bound any occurrences of signature parameters, so, in particular, the scheme $f(f(x)|f(x,y))$ represents exactly the same terms as the scheme $f(f(x))$.

B.2 Sets of Terms and Literals

The scheme notation allows one to define simple patterns of terms. However, its expressive power (i.e. the sets of terms it can define) is rather limited, since, in particular, the number of arguments of every functional parameter has to be explicitly given. It is not possible to represent, say, the set of all ground terms with depth two. For expressing this and other things, we introduce an additional constructor for representing *sets* of expressions. Using this constructor, one can give more flexible specification of possible arguments of an expression or possible literals of a clause.

Given scheme expressions $\boldsymbol{\hat{s}_i}$ with $1 \leq i \leq n$ and $\boldsymbol{\hat{t}_j}$ with $1 \leq j \leq m$ for some $n \geq 0$, $m \geq 0$, the *set scheme* $\boldsymbol{\hat{S}} = \{!\boldsymbol{\hat{s}_1},.., !\boldsymbol{\hat{s}_n}, \boldsymbol{\hat{t}_1},.., \boldsymbol{\hat{t}_m}\}$ represents any set of expressions of the form $S = S_1 \cup \dots \cup S_n \cup T_1 \cup \dots \cup T_m$, where (i) each S_i with $1 \leq i \leq n$ is a *non-empty* set of expressions represented by the scheme $\boldsymbol{\hat{s}_i}$; and (ii) each T_j with $1 \leq j \leq m$ is a (possibly empty) set of expressions represented by the scheme $\boldsymbol{\hat{t}_j}$.

Intuitively, the set scheme $\boldsymbol{\hat{S}}$ represents any set S of expressions, whose elements are represented by one of the schemes $\boldsymbol{\hat{s}_1},.., \boldsymbol{\hat{s}_n}, \boldsymbol{\hat{t}_1},.., \boldsymbol{\hat{t}_m}$ with the additional requirement that there should be at least one representative for every scheme $\boldsymbol{\hat{s}_i}$ with $1 \leq i \leq n$. This is indicated by the *non-emptiness operator* used in front of the scheme expression: $!\hat{\boldsymbol{s}}_{\boldsymbol{i}}$.

For example, the set scheme $\{\hat{c}\}$ represents any set of constants, whereas $\{!\hat{c}\}$ represents any non-empty set of constants. The set scheme $\{!\hat{f}(x), x\}$ in the signature $\Sigma_{\mathcal{A}}$ represents any subset of $\{\mathtt{s}(x), \mathtt{exp}(x), x\}$ containing at least one of the first two terms.

The set operator $\{..\}$ does not bound the free parameters of expressions, so different occurrences of the same free parameter should correspond to the same signature element for every set. According to this, the sets represented by the scheme $\{!\hat{f}(x), f(x,x), x\}$ may additionally contain one the terms $+(x,x)$ or $\times(x,x)$ (but not both of them).

We will use the set schemes for specifying arguments of expressions. Let $\hat{\boldsymbol{T}}$ be a set scheme that represents sets of terms. Then the scheme $f(\hat{\boldsymbol{T}})$ represents the terms, whose set of direct arguments is represented by $\hat{\boldsymbol{T}}$. Continuing the previous example, the scheme $f(\{!\hat{f}(x), x\})$ represents the terms $+(\mathtt{s}(x), x)$, $\mathtt{s}(\mathtt{exp}(x))$, $\times(\mathtt{s}(x), \mathtt{exp}(x))$, but not, say, $\times(x,x)$. A more exotic example can be given by the scheme $f(\{f(x,x), x\})$, which represents the terms $+(x, +(x,x))$, $\mathtt{s}(x)$ and $\times(\times(x,x), \times(x,x))$, but not the terms $\mathtt{s}(+(x,x))$ or $\mathtt{exp}(\mathtt{exp}(x))$. For conciseness we will replace the pair of braces $(\{..\})$ by $[..]$. So the term schemes given above can be shortly written as $f[!\hat{f}(x), x]$ and $f[f(x,x), x]$ respectively.

The set constructor can be also used for specifying literals in clauses. If $\hat{\boldsymbol{L}}$ represents a set of literals $\{l_1,.., l_n\}$, then we write $\vee\hat{\boldsymbol{L}}$ to represent the clause $l_1 \vee \cdots \vee l_n$. For example, the clause scheme $\vee\{!\neg a[x], a[\hat{f}(x,x)]\} \vee a[!\hat{f}(x)]$ represents the clause $\neg\mathtt{Nat}(x) \vee \mathtt{Nat}(\mathtt{s}(x))$, but not the clauses $\mathtt{Nat}(\mathtt{s}(x))$, $\neg\mathtt{Nat}(x) \vee \mathtt{Nat}(+(x,x))$, or $\neg\mathtt{Nat}(x) \vee \mathtt{Nat}(\mathtt{s}(x)) \vee \mathtt{Nat}(\mathtt{exp}(x))$. We usually omit the disjunction symbol before the set expression $\vee\hat{\boldsymbol{L}}$ in clause schemes, when it is clearly used in a clause context. Note, that the clause schemes $\neg a(x) \vee \{!a(f(x))\}$ and $\neg a(x) \vee a(f(x))$ represent essentially the same clause sets (modulo duplicate literals), but $\neg a(x) \vee \{!a(\hat{f}(x))\}$ and $\neg a(x) \vee a(\hat{f}(x))$ don't: the first scheme represents the clause $\neg\mathtt{Nat}(x) \vee \mathtt{Nat}(\mathtt{s}(x)) \vee \mathtt{Nat}(\mathtt{exp}(x))$ but doesn't represent the second scheme.

B.3 Variable-Vectors and Scheme definitions

The set constructor allows one to represent expressions with an unbounded number of arguments. But so far, all variables that may appear in expressions have to be explicitly mentioned in a scheme. Thus, it was not possible to represent, say, the set of shallow terms (terms containing only variables), or terms of the functional depth two (except than enumerating them explicitly).

To represent expressions with unbounded number of variables, we will use variable-vectors. A *variable-vector (parameter)* $\overline{x}$ represents any (possibly empty) sequence

Table B.2 Summary for the usage of the set constructor in schemes

schemes	**represent**	**do not represent,** ***why***
$\{\hat{c}\}$	$\{\mathtt{zero}\},\ \{\}$	
$\{!\hat{c}\}$	$\{\mathtt{zero}\}$	$\{\}$: *non-emptiness operator*
$\hat{f}[c]$		: *ground terms of depth two*
$\{!\hat{f}(x), x\}$	$\{\mathtt{s}(x), x\}$ $\{\mathtt{s}(x), \mathtt{exp}(x)\}$	$\{x\}$: *non-emptiness operator*
$\{!\hat{f}(x), f(x,x), x\}$	$\{\mathtt{exp}(x), +(x,x), x\}$ $\{\mathtt{s}(x), \times(x,x)\}$	$\{\mathtt{exp}(x), +(x,x), \times(x,x), x\}$: *f is a free in the set scheme*
$f[f(x,x), x]$	$+(x, +(x,x)),\ \mathtt{s}(x)$ $\times(\times(x,x), \times(x,x))$	$\mathtt{s}(+(x,x))$: *free parameter f* $\mathtt{exp}(\mathtt{exp}(x))$: *f is a binary symbol*
$\{!\neg a[x], a[\hat{f}(x,x)]\} \vee$ $\vee\, a[!\hat{f}(x)]$	$\neg\mathtt{Nat}(x) \vee \mathtt{Nat}(\mathtt{s}(x))$	$\mathtt{Nat}(\mathtt{s}(x))$: $!\neg a[x]$ $\neg\mathtt{Nat}(x) \vee \mathtt{Nat}(+(x,x))$ $\neg\mathtt{Nat}(x) \vee \mathtt{Nat}(\mathtt{s}(x)) \vee \mathtt{Nat}(\mathtt{exp}(x))$: *the literal for $a[!\hat{f}(x)]$ is unique*
$\neg a(x) \vee a(\hat{f}(x)) \not\equiv$ $\neg a(x) \vee \{!a(\hat{f}(x))\}$	$\neg a(x) \vee a(\mathtt{exp}(x))$ $\neg a(x) \vee a(\mathtt{s}(x))$	$a(x) \vee a(\mathtt{s}(x)) \vee a(\mathtt{exp}(x))$: *first scheme $\Rightarrow$ two-literal clauses*

of (not necessarily disjoint) variables $x_1, x_2,.., x_n$, $n \geq 0$. Variable-vectors can be used as arguments of scheme expressions: $(.., \overline{x},..)$, $[.., \overline{x},..]$, $[.., !\overline{x},..]$ which should expand to $(.., x_1, x_2,.., x_n,..)$, $[.., x_1, x_2,.., x_n,..]$ and $[.., !x_1, !x_2,.., !x_n,..]$ respectively. The variable-vectors are not bounded in schemes, so different occurrences of the same variable-vector should correspond to the same sequence of variables. For example, the scheme $\hat{f}(\overline{x}, \hat{f}(\overline{x}))$ represents the terms $+(x, \mathtt{exp}(x))$ and $\mathtt{s}(\mathtt{zero})$ (with $\overline{x}$ being the empty sequence), but not the terms $+(\mathtt{s}(x), x)$ or $\times(x, +(x,x))$. The scheme $f[!\hat{f}(\overline{x}), \overline{x}]$ represents the terms $+(\mathtt{s}(x), \mathtt{exp}(x))$, $\times(+(x,y), \times(x,y))$, and $\times(x, +(y,x))$ but not $+(\mathtt{zero}, x)$ or $\times(+(x,y), \times(y,x))$. The scheme $f(f(\overline{x}), \overline{x})$ represents no terms.

A variable-vector refers to the same sequence of variables in every position of its occurrence in a scheme. It would be nice to have this property not only for sequences of variables, but also for other scheme expressions. For instance, in some situations we need to represent clauses which have the same sequence of arguments in every functional term. For expressing this sort of conditions, we will use *scheme definitions*. Informally, a scheme definition associates a scheme expression with a *scheme parameter*. This scheme parameter can be used in a scheme instead of the scheme expressions it is assigned to. However, when a scheme parameter is used in a scheme expressions in several positions, all these positions should correspond to the *same value* of the scheme parameter.

For example, we can define a scheme parameter $\boldsymbol{s} := \hat{f}[\overline{x}]$ for shallow terms and use it in a clause scheme $\vee\{\hat{a}[\boldsymbol{s}, \overline{x}], \neg\hat{a}[\boldsymbol{s}, \overline{x}]\}$. This clause scheme represents the set of clauses containing (several occurrences of) at most one functional subterm. If we replace the scheme variable by the correspondent scheme expression it defines: $\vee\{\hat{a}[\hat{f}[\overline{x}],\overline{x}],\neg\hat{a}[\hat{f}[\overline{x}],\overline{x}]\}$, we will capture more clauses, namely all clauses with functional depth at most two.

For another example, define a scheme variable $\boldsymbol{v} := [!\overline{x},\hat{c}]$ representing vectors of variables and constants. We can use this scheme variable in the clause scheme $\vee\{\hat{a}[\hat{f}(\boldsymbol{v}),\overline{x},\hat{c}],\neg\hat{a}[\hat{f}(\boldsymbol{v}),\overline{x},\hat{c}]\}$, which represents the set of clauses whose functional subterms are either constants, or otherwise have a fixed (within a clause) vector of arguments, consisting of all variables of the clause and constants.

The scheme expressions in definitions of scheme parameters may possibly contain other scheme parameters, but we do not admit cyclic definitions. So, both $\boldsymbol{v} := [\hat{f}(\boldsymbol{v}),\overline{x}]$ and $\boldsymbol{s} := \hat{f}[\hat{f}(\boldsymbol{u}),\hat{c}]$; $\boldsymbol{u} := [\boldsymbol{s},\overline{x}]$ are not admissible scheme definitions.

Table B.3 Summary for the usage of variable-vectors and scheme definitions in schemes

schemes	**represent**	**do not represent,**	***why***
$\hat{f}(\overline{x},\hat{f}(\overline{x}))$	$+(x,\mathtt{exp}(x))$, $\mathtt{s}(\mathtt{zero})$.	$+(\mathtt{s}(x),x)$ $\times(x,+(x,x))$	: *the variable should go first* : *all $\overline{x}$ have to be the same vectors*
$f[!\hat{f}(\overline{x}),\overline{x}]$	$+(\mathtt{s}(x),\mathtt{exp}(x))$, $\times(+(x,y),\times(x,y))$, $\times(x,+(y,x))$.	$+(\mathtt{zero},x)$ $\times(+(x,y),\times(y,x))$	: $\hat{f}(\overline{x})$ *contains all variables* : *all $\overline{x}$ have to be the same vectors*
$f(f(\overline{x}),\overline{x})$	(nothing)		: *inconsistent conditions on the arity of f*
$\boldsymbol{s}:=\hat{f}[\overline{x}]$, $\hat{f}[\boldsymbol{s},\overline{x}]$	$+(\mathtt{s}(x),\mathtt{s}(x))$ $\times(\times(x,y),z)$	$+(\mathtt{s}(x),\mathtt{exp}(x))$ $\times(\times(x,y),\times(y,x))$	 : *different functional subterms*
$\boldsymbol{v}:=[!\overline{x},\hat{c}]$, $\hat{f}(\hat{f}(\boldsymbol{v}),\overline{x})$	$+(+(x,y),\times(x,y))$ $\times(x,+(\mathtt{zero},x))$	$+(+(x,y),\times(y,x))$ $\times(y,+(\mathtt{zero},x))$	: *different vectors of arguments* : $+(\mathtt{zero},x)$ *does not contain y*

B.4 The Formal Semantics for Clause Schemes

Table B.4 Recursive definitions for schemes of expressions and clauses

$$\begin{aligned}
\hat{\boldsymbol{t}} &::= x \mid \mathtt{a}(\hat{\boldsymbol{V}}) \mid p(\hat{\boldsymbol{V}}) \mid \hat{p}(\hat{\boldsymbol{V}}) \mid (\hat{\boldsymbol{t}}_{\boldsymbol{1}}|\hat{\boldsymbol{t}}_{\boldsymbol{2}}) \mid \langle\hat{\boldsymbol{t}}\rangle\ .\\
\hat{\boldsymbol{a}} &::= \qquad \mathtt{f}(\hat{\boldsymbol{V}}) \mid p(\hat{\boldsymbol{V}}) \mid \hat{p}(\hat{\boldsymbol{V}}) \mid (\hat{\boldsymbol{a}}_{\boldsymbol{1}}|\hat{\boldsymbol{a}}_{\boldsymbol{2}}) \mid \langle\hat{\boldsymbol{a}}\rangle\ .\\
\hat{\boldsymbol{l}} &::= \hat{\boldsymbol{a}} \mid \neg\hat{\boldsymbol{a}}\ .\\
\hat{\boldsymbol{C}} &::= \square \mid \hat{\boldsymbol{l}} \mid (\vee\hat{\boldsymbol{L}}) \mid (\hat{\boldsymbol{C}}_{\boldsymbol{1}}\vee\hat{\boldsymbol{C}}_{\boldsymbol{2}}) \mid \langle\hat{\boldsymbol{C}}\rangle\ .\\
\hat{\boldsymbol{V}} &::= () \mid (\hat{\boldsymbol{t}}) \mid (\overline{x}) \mid (\hat{\boldsymbol{T}}) \mid (\hat{\boldsymbol{V}}_{\boldsymbol{1}};\hat{\boldsymbol{V}}_{\boldsymbol{2}}) \mid \langle\hat{\boldsymbol{V}}\rangle\ .\\
\hat{\boldsymbol{T}} &::= \{\} \mid \{\hat{\boldsymbol{t}}\} \mid \{!\hat{\boldsymbol{t}}\} \mid \{\overline{x}\} \mid \{!\overline{x}\} \mid (\hat{\boldsymbol{T}}_{\boldsymbol{1}}\cup\hat{\boldsymbol{T}}_{\boldsymbol{2}}) \mid \langle\hat{\boldsymbol{T}}\rangle\ .\\
\hat{\boldsymbol{L}} &::= \{\} \mid \{\hat{\boldsymbol{l}}\} \mid \{!\hat{\boldsymbol{l}}\} \mid (\hat{\boldsymbol{L}}_{\boldsymbol{1}}\cup\hat{\boldsymbol{L}}_{\boldsymbol{2}}) \mid \langle\hat{\boldsymbol{L}}\rangle\ .
\end{aligned}$$

Table B.5 The formal semantics for schemes of expressions and clauses

$$\begin{array}{rcll}
[\hat{\boldsymbol{t}}]^{\eta,\nu}_{\hat{t}} : = [x]^{\eta,\nu}_{\hat{t}} &=& \{x\} & | \\
[\mathtt{f}(\hat{\boldsymbol{V}})]^{\eta,\nu}_{\hat{t}} &=& \{\mathtt{f}([\hat{\boldsymbol{V}}]^{\eta,\nu}_{\hat{V}})\} & | \\
[p(\hat{\boldsymbol{V}})]^{\eta,\nu}_{\hat{t}} &=& \{p^{\eta}([\hat{\boldsymbol{V}}]^{\eta,\nu}_{\hat{V}})\} & | \\
[\hat{p}(\hat{\boldsymbol{V}})]^{\eta,\nu}_{\hat{t}} &=& \cup_{\eta'}\{p^{\eta'}([\hat{\boldsymbol{V}}]^{\eta,\nu}_{\hat{V}})\} & | \\
[(\hat{\boldsymbol{t}}_{\boldsymbol{1}}|\hat{\boldsymbol{t}}_{\boldsymbol{2}})]^{\eta,\nu}_{\hat{t}} &=& [\hat{\boldsymbol{t}}_{\boldsymbol{1}}]^{\eta,\nu}_{\hat{t}} \cup [\hat{\boldsymbol{t}}_{\boldsymbol{2}}]^{\eta,\nu}_{\hat{t}} & | \\
[\langle\hat{\boldsymbol{t}}\rangle]^{\eta,\nu}_{\hat{t}} &=& \cup_{\eta'}[\hat{\boldsymbol{t}}]^{\eta',\nu}_{\hat{t}} \ . \cap \boldsymbol{Tm_{\Sigma}} &
\end{array}$$

$$\begin{array}{rcll}
[\hat{\boldsymbol{a}}]^{\eta,\nu}_{\hat{a}} : = && & \\
[\mathtt{a}(\hat{\boldsymbol{V}})]^{\eta,\nu}_{\hat{t}} &=& \{\mathtt{a}([\hat{\boldsymbol{V}}]^{\eta,\nu}_{\hat{V}})\} & | \\
[p(\hat{\boldsymbol{V}})]^{\eta,\nu}_{\hat{a}} &=& \{p^{\eta}([\hat{\boldsymbol{V}}]^{\eta,\nu}_{\hat{V}})\} & | \\
[\hat{p}(\hat{\boldsymbol{V}})]^{\eta,\nu}_{\hat{a}} &=& \cup_{\eta'}\{p^{\eta'}([\hat{\boldsymbol{V}}]^{\eta,\nu}_{\hat{V}})\} & | \\
[(\hat{\boldsymbol{a}}_{\boldsymbol{1}}|\hat{\boldsymbol{a}}_{\boldsymbol{2}})]^{\eta,\nu}_{\hat{a}} &=& [\hat{\boldsymbol{a}}_{\boldsymbol{1}}]^{\eta,\nu}_{\hat{t}} \cup [\hat{\boldsymbol{a}}_{\boldsymbol{2}}]^{\eta,\nu}_{\hat{t}} & | \\
[\langle\hat{\boldsymbol{a}}\rangle]^{\eta,\nu}_{\hat{a}} &=& \cup_{\eta'}[\hat{\boldsymbol{a}}]^{\eta',\nu}_{\hat{t}} \ . \cap \boldsymbol{At_{\Sigma}} &
\end{array}$$

$$\begin{array}{rcll}
[\hat{\boldsymbol{l}}]^{\eta,\nu}_{\hat{l}} : = && & \\
[\hat{\boldsymbol{a}}]^{\eta,\nu}_{\hat{l}} &=& [\hat{\boldsymbol{a}}]^{\eta,\nu}_{\hat{a}} & | \\
[\neg\hat{\boldsymbol{a}}]^{\eta,\nu}_{\hat{l}} &=& \neg[\hat{\boldsymbol{a}}]^{\eta,\nu}_{\hat{a}} \ . &
\end{array}$$

$$\begin{array}{rcll}
[\hat{\boldsymbol{C}}]^{\eta,\nu}_{\hat{C}} : = && & \\
[\Box]^{\eta,\nu}_{\hat{C}} &=& \Box & | \\
[\hat{\boldsymbol{l}}]^{\eta,\nu}_{\hat{C}} &=& [\hat{\boldsymbol{l}}]^{\eta,\nu}_{\hat{l}} & | \\
[\vee(\hat{\boldsymbol{L}})]^{\eta,\nu}_{\hat{C}} &=& \{\vee\{S\} \,|\, S \in [\hat{\boldsymbol{L}}]^{\eta,\nu}_{\hat{L}}\} & | \\
[\hat{\boldsymbol{C}}_{\boldsymbol{1}} \vee \hat{\boldsymbol{C}}_{\boldsymbol{2}}]^{\eta,\nu}_{\hat{C}} &=& \{C_1 \vee C_2 \,|\, C_i \in [\hat{\boldsymbol{C}}_{\boldsymbol{i}}]^{\eta,\nu}_{\hat{C}}\} & | \\
[\langle\hat{\boldsymbol{C}}\rangle]^{\eta,\nu}_{\hat{C}} &=& \cup_{\eta'}[\hat{\boldsymbol{C}}]^{\eta',\nu}_{\hat{C}} \ . &
\end{array}$$

$$\begin{array}{rcll}
[\hat{\boldsymbol{V}}]^{\eta,\nu}_{\hat{V}} : = [()]^{\eta,\nu}_{\hat{V}} : () && & | \\
[(\hat{\boldsymbol{t}})]^{\eta,\nu}_{\hat{V}} &=& \{(t) \,|\, t \in [\hat{\boldsymbol{t}}]^{\eta,\nu}_{\hat{t}}\} & | \\
[(\overline{x})]^{\eta,\nu}_{\hat{V}} &=& \{\nu(\overline{x})\} & | \\
[(\hat{\boldsymbol{T}})]^{\eta,\nu}_{\hat{V}} &=& \{(t_1,..,t_n) \,|\, \{t_1,..,t_n\} \in [\hat{\boldsymbol{T}}]^{\eta,\nu}_{\hat{T}}\} & | \\
[(\hat{\boldsymbol{V}}_{\boldsymbol{1}};\hat{\boldsymbol{V}}_{\boldsymbol{2}})]^{\eta,\nu}_{\hat{V}} &=& \{(V_1;V_2) \,|\, V_i \in [\hat{\boldsymbol{V}}_{\boldsymbol{i}}]^{\eta,\nu}_{\hat{V}}\} & | \\
[\langle\hat{\boldsymbol{V}}\rangle]^{\eta,\nu}_{\hat{V}} &=& \cup_{\eta'}[\hat{\boldsymbol{V}}]^{\eta',\nu}_{\hat{V}} \ . &
\end{array}$$

$$\begin{array}{rcll}
[\hat{\boldsymbol{T}}]^{\eta,\nu}_{\hat{T}} : = [\{\}]^{\eta,\nu}_{\hat{T}} &=& \{\} & | \\
[\{\hat{\boldsymbol{t}}\}]^{\eta,\nu}_{\hat{T}} &=& \{T \,|\, T \subseteq [\hat{\boldsymbol{t}}]^{\eta,\nu}_{\hat{t}}\} & | \\
[\{!\hat{\boldsymbol{t}}\}]^{\eta,\nu}_{\hat{T}} &=& \{T \,|\, \{\} \neq T \subseteq [\hat{\boldsymbol{t}}]^{\eta,\nu}_{\hat{t}}\} & | \\
[\{\overline{x}\}]^{\eta,\nu}_{\hat{T}} &=& \{T \,|\, T \subseteq \nu(\overline{x})\} & | \\
[\{!\overline{x}\}]^{\eta,\nu}_{\hat{T}} &=& \mathit{Set}(\nu(\overline{x})) & | \\
[\hat{\boldsymbol{T}}_{\boldsymbol{1}} \cup \hat{\boldsymbol{T}}_{\boldsymbol{2}}]^{\eta,\nu}_{\hat{T}} &=& \{T_1 \cup T_2 \,|\, T_i \in [\hat{\boldsymbol{T}}_{\boldsymbol{i}}]^{\eta,\nu}_{\hat{T}}\} & | \\
[\langle\hat{\boldsymbol{T}}\rangle]^{\eta,\nu}_{\hat{T}} &=& \cup_{\eta'}[\hat{\boldsymbol{T}}]^{\eta',\nu}_{\hat{T}} \ . &
\end{array}$$

$$\begin{array}{rcll}
[\hat{\boldsymbol{L}}]^{\eta,\nu}_{\hat{L}} : = [\{\}]^{\eta,\nu}_{\hat{L}} &=& \{\} & | \\
[\{\hat{\boldsymbol{l}}\}]^{\eta,\nu}_{\hat{L}} &=& \{S \,|\, S \subseteq [\hat{\boldsymbol{l}}]^{\eta,\nu}_{\hat{t}}\} & | \\
[\{!\hat{\boldsymbol{l}}\}]^{\eta,\nu}_{\hat{L}} &=& \{S \,|\, \{\} \neq S \subseteq [\hat{\boldsymbol{l}}]^{\eta,\nu}_{\hat{t}}\} & | \\
[\hat{\boldsymbol{L}}_{\boldsymbol{1}} \cup \hat{\boldsymbol{L}}_{\boldsymbol{2}}]^{\eta,\nu}_{\hat{L}} &=& \{S_1 \cup S_2 \,|\, S_i \in [\hat{\boldsymbol{L}}_{\boldsymbol{i}}]^{\eta,\nu}_{\hat{L}}\} & | \\
[\langle\hat{\boldsymbol{L}}\rangle]^{\eta,\nu}_{\hat{L}} &=& \cup_{\eta'}[\hat{\boldsymbol{L}}]^{\eta',\nu}_{\hat{L}} \ . &
\end{array}$$

where $\vee\{L_1,..,L_n\} = L_1 \vee\cdots\vee L_n;$
$i = 1,2;$ $\mathit{Set}(x_1,..,x_n) = \{x_1,..,x_n\}.$

The syntax for *clause schemes* $\hat{\boldsymbol{C}}$ is defined recursively on Table B.4 using *term schemes* $\hat{\boldsymbol{t}}$, *atom schemes* $\hat{\boldsymbol{a}}$, *literal schemes* $\hat{\boldsymbol{l}}$ and other auxiliary schemes: $\hat{\boldsymbol{V}}$ for vectors of terms, $\hat{\boldsymbol{T}}$ for sets of terms and $\hat{\boldsymbol{L}}$ for sets of literals. Here a, f denote predicate and functional symbols, and p is a signature parameter. Note, that for simplicity we have restricted the application of the bounding operator. It can be applied either to a parameter, or to a well-formed scheme expression.

The semantics of scheme expressions is given in terms of a parameter assignment η and a variable-vector assignment ν. A *(signature) parameter assignment* η is a function that assigns a signature element to every signature parameter according to its range. For example, for the parameter definitions given before, $\eta : f \rightarrow \{+, \times, \mathtt{s}, \mathtt{exp}, \mathtt{zero}\}$, $\eta : a \rightarrow \{\mathtt{Nat}\}$, $\eta : f_2 \rightarrow \{+, \times\}$, $\eta : f_1 \rightarrow \{\mathtt{s}, \mathtt{exp}\}$, $\eta : c \rightarrow \{\mathtt{zero}\}$. The value of a parameter p under a parameter assignment η is denoted by p^{η}. A *variable-vector assignment* ν is a function that assigns a vector of variables to every variable-vector parameter. For example, ν can map the variable-vector $\overline{x}$ to the vector $\nu(\overline{x}) = (x, y, x, x)$, or to the *empty vector* $\nu(\overline{x}) = ()$.

The value of a clause scheme $\hat{C} \in \hat{\boldsymbol{C}}$ under a parameter assignment η and a variable-vector assignment ν is a clause set $[\hat{C}]^{\eta,\nu}_{\hat{\boldsymbol{C}}}$, where the function $[\hat{\boldsymbol{C}}]^{\eta,\nu}_{\hat{\boldsymbol{C}}}$ is defined on Tab. B.5 by mutual recursion using the functions $[\hat{\boldsymbol{t}}]^{\eta,\nu}_{\hat{\boldsymbol{t}}}$, $[\hat{\boldsymbol{a}}]^{\eta,\nu}_{\hat{\boldsymbol{a}}}$, $[\hat{\boldsymbol{l}}]^{\eta,\nu}_{\hat{\boldsymbol{l}}}$ that map a term scheme, an atom scheme or a literal scheme to a set of terms, literals and clauses respectively; and the functions $[\hat{\boldsymbol{V}}]^{\eta,\nu}_{\hat{\boldsymbol{V}}}$, $[\hat{\boldsymbol{T}}]^{\eta,\nu}_{\hat{\boldsymbol{T}}}$, $[\hat{\boldsymbol{L}}]^{\eta,\nu}_{\hat{\boldsymbol{L}}}$ that map a scheme expression to a set of term vectors, term sets and literal sets respectively. Note, that a value of a bounded scheme expression $[\langle\hat{s}\rangle]^{\eta,\nu}_{*}$ is the union of its values for all parameter assignments $\cup_{\eta'}[\hat{s}]^{\eta',\nu}_{*}$, therefore, it does not depend on η, as expected. Here $\boldsymbol{Tm_\Sigma}$ and $\boldsymbol{At_\Sigma}$ are the sets of terms and atoms over the signature Σ. As usual, $\cup$ is a union operator for sets and ; is a concatenation operator for lists.

To evaluate scheme expressions involving scheme parameters (defined by other scheme expressions) the above definitions should be modified as follows. Suppose, $\hat{\boldsymbol{s}}$ is a scheme containing free occurrences of scheme parameters $\boldsymbol{p_1}, .., \boldsymbol{p_n}$, $n \geq 1$ defined by $\boldsymbol{p_i} := \hat{\boldsymbol{s}}_{\boldsymbol{i}}$, $i = 1 \ldots n$. Given a parameter assignment η and a variable-vector assignment ν, the value of the scheme $\hat{\boldsymbol{s}}$ is defined by inductively as

$$[\hat{\boldsymbol{s}}]^{\eta,\nu}_{*} := \bigcup_{s_1 \in [\hat{\boldsymbol{s}}_{\boldsymbol{1}}]^{\eta,\nu}_{*},..,s_n \in [\hat{s_n}]^{\eta,\nu}_{*}} [\hat{\boldsymbol{s}}[\boldsymbol{p_1}/s_1,.., \boldsymbol{p_n}/s_n]]^{\eta,\nu}_{*},$$

that is, the union of the values of the scheme over all substitution of free scheme parameters by its values under η and ν (where different occurrences of the same parameter are evaluated by the same expression).

B.5 Scheme Contexts and Defined Parameters

The scheme language described above is sufficient for representing most of our clause classes and inference cases. In this section we introduce additional abbreviations that allow us to write clause schemes even more compactly. For example, it would be possible to shorten, say, the clause scheme $\vee\{\hat{p}[!\hat{f}(\overline{x}), \overline{x}], \neg\hat{p}[!\hat{f}(\overline{x}), \overline{x}]\}$ to $\vee\{\neg\hat{p}, \hat{p}\}[!\hat{f}(\overline{x}), \overline{x}]$ and even to $\hat{\alpha}[!\hat{f}(\overline{x}), \overline{x}]$ by defining a *clause context parameter* $\alpha := \vee\{\neg\hat{p}, \hat{p}\}$.

An *expression context* is either a functional symbol (constant), a predicate symbol or a negated predicate symbol. Loosely speaking, an expression context is an expression without its arguments. If $E = \{e_1,.., e_k\}$ is a set of expression contexts and $(t_1,.., t_n)$ is a vector of arguments, $k \geq 0, n \geq 0$, then $E(t_1, \ldots, t_n)$ denotes the set of expressions $\{e_1(t_1, \ldots, t_n),..e_k(t_1, \ldots, t_n)\}$. Similarly, we can attach scheme arguments $(\hat{\boldsymbol{t}}_{\boldsymbol{1}},.., \hat{\boldsymbol{t}}_{\boldsymbol{n}})$ or $[!\hat{\boldsymbol{t}}_{\boldsymbol{1}},.., !\hat{\boldsymbol{t}}_{\boldsymbol{n}}, \hat{\boldsymbol{t}}_{\boldsymbol{n+1}}..\hat{\boldsymbol{t}}_{\boldsymbol{n+m}}]$, $n \geq 0$, $m \geq 0$ to a *scheme context* of the form $\hat{\boldsymbol{E}} = \{!\hat{\boldsymbol{e}}_{\boldsymbol{1}},.., !\hat{\boldsymbol{e}}_{\boldsymbol{k}}, \hat{\boldsymbol{e}}_{\boldsymbol{k+1}},.., \hat{\boldsymbol{e}}_{\boldsymbol{k+s}}\}$ where $\hat{\boldsymbol{e}}_{\boldsymbol{i}}$ are (possibly bounded) signature parameters or negated signature parameters, $k \geq 0$, $s \geq 0$. The schemes $\hat{\boldsymbol{E}}(\cdot)$ and $\hat{\boldsymbol{E}}[\cdot]$ should expand to $\{!\hat{\boldsymbol{e}}_{\boldsymbol{1}}(\cdot),.., !\hat{\boldsymbol{e}}_{\boldsymbol{k}}(\cdot), \hat{\boldsymbol{e}}_{\boldsymbol{k+1}}(\cdot),.., \hat{\boldsymbol{e}}_{\boldsymbol{k+s}}(\cdot)\}$ and $\{!\hat{\boldsymbol{e}}_{\boldsymbol{1}}[\cdot],.., !\hat{\boldsymbol{e}}_{\boldsymbol{k}}[\cdot], \hat{\boldsymbol{e}}_{\boldsymbol{k+1}}[\cdot],.., \hat{\boldsymbol{e}}_{\boldsymbol{k+s}}[\cdot]\}$ respectively. We will allow to form new scheme contexts using the same operations as for schemes expressions. These context expressions attached to scheme arguments should expand similarly.

To avoid repetition of the same context expressions, we introduce labels for them called *context parameters*. Context parameters in schemes should be simply expanded to the correspondent expression which they define. For instance, we can define a *literal parameter* $l := (p|\neg p)$ and use it in the scheme $l[\overline{x}]$, that should be expanded to $(p[\overline{x}]|\neg p[\overline{x}])$. Note that according to this definition, different free occurrences of the same context parameters do not necessary correspond to the same expression. For instance, the clause scheme $l(\overline{x}) \vee l(\overline{x})$ is expanded to $(p(\overline{x})|\neg p(\overline{x})) \vee (p(\overline{x})|\neg p(\overline{x}))$ which represent, say, the clause $\mathtt{Nat}(x) \vee \neg\mathtt{Nat}(y)$. In order to avoid confusions caused by this, we will refrain from using free occurrences of context parameters.[1]

Table B.6 Summary for the usage of context parameters in schemes

$\alpha := \{a, \neg a\}$; $k := (b|\neg b)$

schemes	$\Rightarrow$	expand to
$\hat{\alpha}[x] \vee \hat{\alpha}[y]$	$\Rightarrow$	$\{\hat{a}, \neg\hat{a}\}[x] \vee \{\hat{a}, \neg\hat{a}\}[y]$;
$\vee\neg\{!a, \hat{b}\}[f(\overline{x}), x]$	$\Rightarrow$	$\vee\{!\neg a[f(\overline{x}), x], \neg\hat{b}[f(\overline{x}), x]\}$;
$\vee\neg\{!\hat{k}, \neg a\}[\hat{c}]$	$\Rightarrow$	$\vee\{!(\hat{b}[\hat{c}]\mid\neg\hat{b}[\hat{c}]), \neg a[\hat{c}]\}$;

B.6 Indexing of Signature Elements and Parameters

For certain situations we need to use predicate or functional symbols that are indexed by other syntactical objects (signature elements, predicates, literals etc.). For example, when skolemizing the expression $\forall\overline{x}.\exists y_1..y_n.\mathtt{a}(y_1,.., y_n, \overline{x})$, $n \geq 0$, we would like to "remember" that the Skolem functions introduced for variables $y_1,.., y_n$ came from the one sequence of existential quantifiers. This can be achieved by skolemizing the above expression ad follows: $\forall\overline{x}.\mathtt{a}(\mathtt{f}_1^{\mathtt{p}}(\overline{x}),.., \mathtt{f}_n^{\mathtt{p}}(\overline{x}), \overline{x})$, where $\mathtt{p}$ is a special label (index) introduced for the position below the last existential quantifier in the sequence, and $\mathtt{f}_1^{\mathtt{p}},.., \mathtt{f}_n^{\mathtt{p}}$ are Skolem functions *indexed by* $\mathtt{p}$. The skolemized formula correspond to the clause $\mathtt{a}(\overline{x}, \mathtt{f}_1^{\mathtt{p}}(\overline{x}),..\mathtt{f}_n^{\mathtt{p}}(\overline{x}))$.

[1] It is possible to give a semantics, where different free occurrences of the defined parameters correspond to the same expression context. However this is not trivial to do. For example, one can define the parameter $\varphi := \{\hat{f}\}$ that represents sets of functional symbols. Then the semantics for the scheme $\varphi(\varphi[\overline{x}])$ is rather difficult to give, when all occurrences of φ denote the same set. Hopefully, we will not need this feature in our proofs.

To preserve this structural information in clause schemes we use indexed parameters. The above clause can be represented by the clause scheme $a[\hat{f}^p(\overline{x}), \overline{x}]$, where *index parameter* p is free. For any choice of this parameter $p := \mathtt{p}$, the *indexed parameter* $\hat{f}^p$ represent the set of functional symbols indexed by $\mathtt{p}$.

B.7 Conclusions

Table B.7 Summary for the usage of clause schemes

$\Sigma = (\{\mathtt{a}^3, \mathtt{b}^2, \mathtt{T}^2, \simeq^2, \mathtt{u}^1, \mathtt{p}^0\}, \{\mathtt{f}^2, \mathtt{g}^2, \mathtt{h}^1, \mathtt{c}^0, \mathtt{d}^0\})$

$a ::= \mathtt{a}|\mathtt{b}|\mathtt{p}|\simeq|\mathtt{u};\quad T ::= \mathtt{T};\quad f ::= \mathtt{f}|\mathtt{g}|\mathtt{h}|\mathtt{c}|\mathtt{d};\quad c ::= \mathtt{c}|\mathtt{d};\quad \alpha := \{a, \neg a\};\quad l := (a|\neg a)$

schemes	**represent**	**do not represent,** ***why***
$\neg(a\vert T)[\overline{x}] \vee \hat{\alpha}[\hat{f}(\overline{x}), \overline{x}]$	$\neg\mathtt{a}(x, y, x) \vee \mathtt{b}(y, \mathtt{f}(x, y))$, $\neg\mathtt{T}(x, y) \vee \neg\mathtt{a}(y, x)$, $\neg\mathtt{p} \vee \neg\mathtt{b}(\mathtt{c}_1, \mathtt{c}_2)$.	$\neg\mathtt{a}(x, y, x) \vee \mathtt{b}(\mathtt{f}(\underline{x, y}), \mathtt{f}(\underline{y, x}))$ $\mathtt{a}(x, y) \vee \neg\mathtt{T}(y, \mathtt{f}(x, y))$. : *no literal for* $\neg(a\vert T)[\overline{x}]$
$\hat{\alpha}(\overline{x}, f(\overline{x})) \vee \neg\{a, T\}(\overline{x})$	$\neg\mathtt{a}(x, y, \mathtt{f}(x, y)) \vee x \not\simeq y$, $\neg\mathtt{b}(x, y) \vee \neg\mathtt{T}(x, y)$, $\mathtt{u}(c) \vee \neg\mathtt{p}$, $x \simeq \mathtt{f}(x) \vee \neg b(x, \mathtt{f}(x))$, $\square$.	$\neg\mathtt{a}(x, y, \underline{\mathtt{f}}(x, y)) \vee \mathtt{a}(x, y, \underline{\mathtt{g}}(x, y))$, $\neg\underline{\mathtt{b}}(x, y) \vee x \underline{\not\simeq} y$, : *a is free* $\neg\mathtt{a}(x, y, \mathtt{f}(\underline{x, y})) \vee \mathtt{a}(x, x, \mathtt{f}(x, x))$, $\neg\underline{\mathtt{T}}(x, \mathtt{h}(x))$, : $\neg\mathtt{T} \notin \alpha$ $\mathtt{u}(\underline{\mathtt{c}}) \vee \mathtt{u}(\underline{\mathtt{d}}) \vee \underline{\mathtt{p}}$.

The scheme notation allows one to define sets of expressions and clauses. There are many other possibilities to represent sets of expressions (for instance, tree automata or context-free grammars), so why we have not used those methods? Unfortunately almost nothing from these representations fits to our purposes. The reasons are (**1**) the clause language is already involved: it subsumes lists (for defining the arguments of expressions) and multisets (for defining sets of literals); (**2**) there seems to be no regularity in the sets that we define: we would like, for instance, to capture expressions, with equal arguments, but (**3**) the sets of clauses defined by our scheme are usually finite for a given (finite) signature (this is the main argument used in decidability proofs), thus every formalism defining infinite sets of terms would be too overkill.

As has been mentioned, the main purpose of the scheme notation is to present clause classes and saturation inferences for particular classes of formulas in a compact form. However, the scheme notation opens perspectives in defining a language, where saturation-based decision procedures can be specified in terms of the clause classes and saturation strategies similar to those that will be given in the next sections. It might be possible to develop automated tools (that implement the calculus lifted on the scheme level), using which those procedures can be formally verified. So, our scheme notation can be considered as a step towards a framework for formal development of saturation-based decision procedures.

Appendix C

Complexity of Saturation-Based Decision Procedures

In this appendix we give details of complexity calculations for saturation-based decision procedures formulated in chapter 4.

C.1 Resolution-Based Decision Procedures

C.1.1 Complexity of the Procedure for $\mathcal{GF}$

In this section we provide a detailed complexity analysis for the resolution-based procedure deciding the *guarded fragment without equality* described in subsection 4.1.1, Table 4.3.

A direct estimation of space complexity

In the literature on resolution decision procedures [Ganzinger & de Nivelle, 1999; de Nivelle & de Rijke, 2003], complexity estimation is usually done by computing the maximal number of clauses from the considered clause class that might be constructed over the signature of the input clauses. This number bounds the number of clauses that can be generated during saturation and allows one to establish space and time complexity for a decision procedure. We will show, that although this approach gives us an optimal 2EXPTIME complexity for the full guarded fragment $\mathcal{GF}$, it does not give a better complexity for its bounded-variable version $\mathcal{GF}^k$, which is "only" EXPTIME-complete (see [Grädel, 1999]). Hopefully this is only a drawback of this rough calculation. After additional observations we will demonstrate that our saturation-based decision procedure has an optimal complexity for $\mathcal{GF}^k$ as well.

Let $F \in \mathcal{GF}$ be a guarded formula. We estimate the maximal number of normalised clauses from (**G**) that can appear in a saturation for F in terms of:

$\boldsymbol{n}$ - $:= |F|$, *the size of* F;

$\boldsymbol{w}$ - $:= width(F)$, *the width of* F;

$\boldsymbol{f}$ - *the number of different functional symbols (including constants) in* **CNF** *for* F; (C.1)

$\boldsymbol{p}$ - *the number of different predicate symbols in* **CNF** *for* F;

$\boldsymbol{a}$ - *the maximal arity of predicate symbols in* **CNF** *for* F.

By Proposition 4.1.3, after **CNF**-transformation for F, we obtain at most $\boldsymbol{n}$ clauses, each having at most 3 literals and at most $\boldsymbol{w}$ different variables. For the number of different functional symbols in these clauses, we have $\boxed{\boldsymbol{f} < \boldsymbol{n}}$, since each functional symbol is a Skolem function or a Skolem constant that corresponds to an existentially quantified or free variable in F. Similarly, $\boxed{\boldsymbol{p} < 2{\cdot}\boldsymbol{n}}$, since both the number of initial and the number of introduced predicate symbols are strictly bounded by $\boldsymbol{n}$. For $\boldsymbol{a}$, we have $\boxed{\boldsymbol{a} \leq \boldsymbol{n} - 1}$. Below we give an estimation for the maximal number of clauses of forms 1 and 2 from (**G**) without duplicate literals, that can be constructed over this signature in terms of $\boldsymbol{n}$ and $\boldsymbol{w}$.

The number of clauses of form 1: The number of ground atoms, that can be constructed in our signature is bounded by $\boldsymbol{A} := \boldsymbol{p}{\cdot}\boldsymbol{f}^{\boldsymbol{a}} < 2{\cdot}\boldsymbol{n}^{\boldsymbol{n}}$ which is a product of different choices for *a predicate symbol* × *a vector of constants of length* $\boldsymbol{a}$. So, the number of different non-tautological clauses without duplicate literals of form 1 from (**G**) that can be constructed from these atoms is bounded by $\boxed{\boldsymbol{c}_1 := 3^{\boldsymbol{A}} < 3^{2\cdot\boldsymbol{n}^{\boldsymbol{n}}}}$: each atom may either occur positively, or negatively or do not occur in a clause.

The number of clauses of form 2: In order to estimate the number of clauses of form 2 from (**G**), let us fix a sequence of variables $\overline{x} = x_1,..,x_k$, $k \leq \boldsymbol{n}$ consisting of at most $\boldsymbol{w}$ *different* variables. Each atom of form $p[\hat{f}(\overline{x}), \overline{x}]$, has at most $\boldsymbol{a}$ argument positions which can be filled with at most $\boldsymbol{f}$ different functional terms of form $\hat{f}(\overline{x})$ and at most $\boldsymbol{w}$ different variables from $\overline{x}$. So, the number of such atoms for a fixed $\overline{x}$ is bounded by $\boldsymbol{A}_{\overline{x}} := \boldsymbol{p}{\cdot}(\boldsymbol{f} + \boldsymbol{w})^{\boldsymbol{a}} < 2{\cdot}(\boldsymbol{n} + \boldsymbol{w})^{\boldsymbol{n}}$. And the number of clauses that can be constructed from them is bounded by $\boldsymbol{c}_2^{\overline{x}} = 3^{\boldsymbol{A}_{\overline{x}}} < 3^{2\cdot(\boldsymbol{n}+\boldsymbol{w})^{\boldsymbol{n}}}$. The number of different sequences of variables $\overline{x}$ of the length smaller or equal than $\boldsymbol{n}$ constructed from at most $\boldsymbol{w}$ variables is bounded by $\boldsymbol{v} = \sum_{k \leq \boldsymbol{n}} \boldsymbol{w}^k \leq \boldsymbol{n}{\cdot}\boldsymbol{w}^{\boldsymbol{n}}$. So, the total number of normalised clauses of from 2 from (**G**) is bounded by $\boxed{\boldsymbol{c}_2 = \boldsymbol{v}{\cdot}\boldsymbol{c}_2^{\overline{x}} < \boldsymbol{n}{\cdot}\boldsymbol{w}^{\boldsymbol{n}}{\cdot}3^{2\cdot(\boldsymbol{n}+\boldsymbol{w})^{\boldsymbol{n}}}}$.

Summing up these numbers, we obtain a bound on the number of normalised clauses from (**G**) that are *relevant* to F:

$$\boxed{\boldsymbol{c} \;=\; \boldsymbol{c}_1+\boldsymbol{c}_2 \;<\; 3^{2\cdot\boldsymbol{n}^{\boldsymbol{n}}} + \boldsymbol{n}\cdot\boldsymbol{w}^{\boldsymbol{n}}\cdot 3^{2\cdot(\boldsymbol{n}+\boldsymbol{w})^{\boldsymbol{n}}} \;=\; \boxed{2^{2^{O(\boldsymbol{n}\cdot\log\boldsymbol{n})}}}} \tag{C.2}$$

As seen from this estimation, there are at most doubly exponential number of normalised clauses that can be generated by a resolution-based theorem prover for a given guarded formula. This means that the guarded fragment can be decided in doubly exponential space and, as will be seen later, in doubly exponential time. So, resolution yields an optimal decision procedure for $\mathcal{GF}$. However, if we are now concerned with a bounded-variable variant of the guarded fragment $\mathcal{GF}^k$ (which consists of guarded formulas whose width is bounded by k), we see, that the calculations above do not give us complexity better than 2EXPTIME, because the *size* $\boldsymbol{n}$ of a formula F *but not its width* $\boldsymbol{w}$ is on the top of the tower of exponent in estimation (C.2). This parameter originates from the maximal arity $\boldsymbol{a}$ of a predicate symbol. It seems that nothing can be improved, since $\boldsymbol{a}$ can be as large as $\boldsymbol{n}-1$ in a guarded formula even with bounded width: take just an atom $p(x,\ldots,x)$ with $\boldsymbol{n}-1$ occurrences of the *same* variable x.

It is, in principle, possible to translate a formula from $\mathcal{GF}^k$ in a satisfiability-preserving way to another formula from $\mathcal{GF}^k$, which has only symbols of the arity smaller than k (such technique has been used, for instance in [Grädel, Kolaitis & Vardi, 1997] for the *two-variable fragment*). However we show that our procedure has already the best known complexity for $\mathcal{GF}^k$, even without additional transformations. Below we present a different estimation for the number of clauses $\boldsymbol{c}$ which shows that our the decision procedure restricted to $\mathcal{GF}^k$ can be implemented to run in time $2^{O(\boldsymbol{n})}$.

A more accurate analysis of space complexity

Our calculations are now based on the following key observations about the initial clauses that are resulted from **CNF**-transformation for a guarded formula $F \in \mathcal{GF}$ (see Table 4.1) and about resolution inferences for guarded clauses (**G**) shown in Table 4.3. As usual, let $\boldsymbol{n} := |F|$ and $\boldsymbol{w} := width(F)$.

Observation 1: *All literals in a conclusion of* $\mathcal{OR}^{\succ}_{\mathrm{Sel}}$*-inferences are instances of literals in premises of these inferences. Therefore, all literals that may appear in saturation are instances of some literals from the initial clauses.*

Observation 2: *The set of functional symbols (including constants) of the initial clauses from Table 4.1* Fun *can be partitioned into pairwise disjoint sets:* $\mathrm{Fun} = \mathrm{Fun}_1 \sqcup \cdots \sqcup \mathrm{Fun}_r$ *such that* ($\boldsymbol{i}$) *all functional symbols occurring in one clause* *belong to some* Fun_i *with* $1 \leq i \leq r$ *and* (***ii***) *each set* Fun_i *with* $1 \leq i \leq r$, *contains at most* $\boldsymbol{w}$ *functional symbols, i.e.,* $|\mathrm{Fun}_i| \leq \boldsymbol{w}$.

Observation 3: *Two guarded functional clauses from* (**G**) *can be resolved only if they contain a common functional symbol. By Observation 2, this implies that for every clause in saturation, the set of its functional symbols is contained in some* Fun_i *with* $1 \leq i \leq r$.

Summarising these observations, each clause from (**G**) that appears in saturation for F, consists only from instances of literals from the input clauses and may contain at most $\boldsymbol{w}$ different functional symbols from one of the set Fun_i with $1 \leq i \leq r \leq \boldsymbol{n}$. Now we repeat calculations for number of clauses of forms 1 and 2 from (**G**), by taking into account this new information.

The number of clauses of form 1: The number of ground clauses that have the above property can be estimated in the following way. Let us fix a set Fun_i with $1 \leq i \leq r$ and compute the number of ground clauses containing constants only from Fun_i. The literals of these clauses are ground instances of the initial literals. We have at most $\boldsymbol{n}$ initial clauses, each having at most 3 literals, and containing at most $\boldsymbol{w}$ variables. So, the number of ground literals with at most $\boldsymbol{w}$ constants from Fun_i that can be constructed from them is bounded by $\boldsymbol{L}_i \leq 3{\cdot}\boldsymbol{n}{\cdot}\boldsymbol{w}^{\boldsymbol{w}}$. The number of ground clauses for the set Fun_i is bounded by $\boldsymbol{c}_1^i = 2^{\boldsymbol{L}_i} \leq 2^{3\boldsymbol{n}\cdot\boldsymbol{w}^{\boldsymbol{w}}}$. Summing up all these values for all sets Fun_i with $1 \leq i \leq r \leq \boldsymbol{n}$, we obtain that the number of clauses of form 1 is bounded by $\boxed{\boldsymbol{c}_1 \leq \boldsymbol{n}{\cdot}2^{3\boldsymbol{n}\cdot\boldsymbol{w}^{\boldsymbol{w}}}}$.

The number of clauses of form 2: Let us fix a sequence of variables $\overline{x}$ consisting of at most $\boldsymbol{w}$ different variables, and a set of functional symbols Fun_i with $1 \leq i \leq r$. We estimate the number of instances of the initial literals of form $l[\hat{f}(\overline{x}), \overline{x}]$ where all functional symbols are from Fun_i. We have at most $3\boldsymbol{n}$ initial literals, each with at most $\boldsymbol{w}$ variables which can be instantiated by either one of $\boldsymbol{w}$ variables from $\overline{x}$, or by one of $\boldsymbol{w}$ terms $f(\overline{x})$, where $f \in \mathrm{Fun}_i$. So the number of such literals is bounded by $\boldsymbol{L}_{i,\overline{x}} \leq 3{\cdot}\boldsymbol{n}{\cdot}(\boldsymbol{w} + \boldsymbol{w})^{\boldsymbol{w}}$, and the number of clauses of form 2 constructed from them is bounded by $\boldsymbol{c}_2^{i,\overline{x}} = 2^{\boldsymbol{L}_{i,\overline{x}}} \leq 2^{3\boldsymbol{n}\cdot(2\boldsymbol{w})^{\boldsymbol{w}}}$. Summing up these values for all Fun_i, and all $\overline{x}$, we obtain that the number of clauses of form 2 is bounded by $\boxed{\boldsymbol{c}_2 \leq \boldsymbol{n}{\cdot}(\boldsymbol{n}{\cdot}\boldsymbol{w}^{\boldsymbol{n}}){\cdot}2^{3\boldsymbol{n}\cdot(2\boldsymbol{w})^{\boldsymbol{w}}}}$.

Now, our calculations yield a much better bound than (C.2) for the maximal number of normalised clauses from (**G**) that can be produced in saturation for a formula F:

$$\boxed{\boldsymbol{c} \;=\; \boldsymbol{c}_1 + \boldsymbol{c}_2 \;\leq\; \boldsymbol{n}{\cdot}2^{3\boldsymbol{n}\cdot\boldsymbol{w}^{\boldsymbol{w}}} \;+\; \boldsymbol{n}{\cdot}(\boldsymbol{n}{\cdot}\boldsymbol{w}^{\boldsymbol{n}}){\cdot}2^{3\boldsymbol{n}\cdot(2\boldsymbol{w})^{\boldsymbol{w}}} \;\leq\; \boxed{2^{\boldsymbol{n}\cdot 2^{\boldsymbol{w}\cdot(\log \boldsymbol{w}+\epsilon)}}}} \tag{C.3}$$

where ϵ is some constant. One can see how complexity is parametrized w.r.t. the size and the width of a formula: the number of clauses generated by resolution is doubly exponential in the *width* of the input formula and “only” exponential in its *size*. This means that for guarded formulas with bounded width (i.e. from $\mathcal{GF}^k$), the procedure generates only $2^{O(\boldsymbol{n})}$ clauses.

Time complexity

To obtain a time complexity bound for the resolution decision procedure given in Table 4.1, we make use of formula (3.24) derived 3.5.6. First, we have to estimate the size $|N|$ of the input clause set, the maximal size $\boldsymbol{m}$ of a normalised clause from $(\mathbf{G})$, the maximal number $\boldsymbol{s}$ of clauses in a normalised clause set, and the maximal number $\boldsymbol{k}$ of premises in an inference rule of the calculus.

Since all inferences and simplification rules in $\mathcal{OR}^{\succ}_{\mathit{Sel}}$ have at most two premises, we have $\boldsymbol{k} \leq 2$. The value of $\boldsymbol{s}$ is bounded by $\boldsymbol{c}$, which has been derived in (C.3). The size of a clause can be estimated as follows. Each literal in clauses from $(\mathbf{G})$ has either form $l[\hat{c}]$ or $l[\hat{f}(\overline{x}), \overline{x}]$, so its size is bounded by $O(\boldsymbol{a}^2)$. The number of such literals in each clause is bounded by $\boldsymbol{L}_i \leq 3\boldsymbol{n}{\cdot}(2\boldsymbol{w})^{\boldsymbol{w}}$: see calculations for the number of clauses above. Hence the maximal size of a clause $\boldsymbol{m} \leq O(\boldsymbol{a}^2){\cdot}3\boldsymbol{n}{\cdot}(2\boldsymbol{w})^{\boldsymbol{w}} \leq \boldsymbol{n}^3{\cdot}2^{\boldsymbol{w}\cdot(\log \boldsymbol{w}+\epsilon)}$, where ϵ is some constant. The size of the initial clause set can be estimated by $|N| \leq 3{\cdot}2\boldsymbol{n}{\cdot}O(\boldsymbol{a}^2) \leq O(\boldsymbol{n}^3)$ (see Proposition 4.1.3). Summarising, from (C.3) we obtain the following bound on the running time of our procedure:

$$\boxed{\boldsymbol{t} \;\leq\; p(O(\boldsymbol{n}^3)) + \boldsymbol{c} \cdot p(\boldsymbol{n}^3{\cdot}2^{\boldsymbol{w}\cdot(\log \boldsymbol{w}+\epsilon)}{\cdot}\boldsymbol{c}) \;\leq\; \boxed{2^{\boldsymbol{n}\cdot 2^{\boldsymbol{w}\cdot(\log \boldsymbol{w}+\epsilon)}}}} \tag{C.4}$$

where ϵ is some constant. So, the time complexity for our decision procedure, as expected, is asymptotically the same as its space complexity.

C.1.2 Complexity of the Procedure for $\mathcal{FO}^2$

In this section we provide a detailed complexity analysis for the resolution-based procedure deciding the *two-variable fragment without equality* described in subsection 4.1.2, Table 4.6.

Space complexity

We estimate the number of clauses from (**T**) that can be generated for the *two-variable fragment*, similarly as it has been done for the *guarded fragment*. Let $T \in \mathcal{FO}^2$ be a two-variable formula and $\boldsymbol{n}$, $\boldsymbol{w}$, $\boldsymbol{f}$, $\boldsymbol{p}$ and $\boldsymbol{a}$ be parameters for this formula defined as in (C.1) on p. 237. Note that $\boxed{\boldsymbol{w} \leq 2}$, and similar to the guarded fragment, $\boxed{\boldsymbol{f} < \boldsymbol{n}}$, $\boxed{\boldsymbol{p} < 2 \cdot \boldsymbol{n}}$ and $\boxed{\boldsymbol{a} \leq \boldsymbol{n} - 1}$. The following observation is analogous to observations given for the guarded fragment:

Observation 1: *All literals in clauses derived from the input clauses for T according to the strategy described in Table 4.6, are either instances of some literals occurring in the input clauses, or a literal of form $(\neg)p_L(x)$, or its instances, where L is an instance of some literal contained in the initial clauses.*

The number of clauses of form 1: First, we estimate the number of ground clauses of form 1 from (**T**). We have at most $3 \cdot \boldsymbol{n}$ different literals in the initial clauses, each containing at most two different variables. Hence the Literal Projection rule can introduce at most $6 \cdot \boldsymbol{n}$ new literals with one variable. So the number of literals of form $l[\mathsf{c}_1, \mathsf{c}_2]$ is bounded by $\boldsymbol{L}_1 \leq 3\boldsymbol{n} \cdot 2^2 + 6\boldsymbol{n} \cdot 2^1 = O(\boldsymbol{n})$, and the number of clauses of form 1 is bounded by $\boxed{\boldsymbol{c}_1 \leq 2^{\boldsymbol{L}_1} \leq 2^{O(\boldsymbol{n})}}$.

The number of clauses of form 2: The estimation is exactly the same as in the previous case, the only difference is that the variables of the relevant literals are instantiated not with constant, but with other two variables: $\boxed{\boldsymbol{c}_2 \leq 2^{\boldsymbol{L}_2} \leq 2^{O(\boldsymbol{n})}}$

The number of clauses of form 3: If we fix the functional term $f(x)$ that occurs in a clause of form 3 then the number of such clauses is estimated exactly as in the previous cases: $\boldsymbol{c}_{3,f} \leq 2^{\boldsymbol{L}_{3,f}} \leq 2^{O(\boldsymbol{n})}$. Summing up these numbers for all functional symbols, we obtain: $\boxed{\boldsymbol{c}_3 \leq \boldsymbol{f} \cdot 2^{O(\boldsymbol{n})}}$.

So the maximal number of clauses that can be generated by our strategy for the two-variable fragment is bounded by:

$$\boxed{\boldsymbol{c} \ = \ \boldsymbol{c}_1 + \boldsymbol{c}_2 + \boldsymbol{c}_3 \ \leq \ 2^{O(\boldsymbol{n})} + 2^{O(\boldsymbol{n})} + \boldsymbol{f} \cdot 2^{O(\boldsymbol{n})} \ \leq \ \boxed{2^{O(\boldsymbol{n})}}} \tag{C.5}$$

Time complexity

Time complexity of our procedure is estimated according to the general formula (3.24) derived in 3.5.6 on p. 92 similarly, as it has been done for the guarded fragment. The required parameters can be estimated as follows: the size of the input clause set $|N| \leq O(\boldsymbol{n})$, the maximal size of a normalised clause set from (**T**) is $\boldsymbol{s} \leq 2^{O(\boldsymbol{n})}$, the maximal size of a normalised clause is $\boldsymbol{m} \leq O(\boldsymbol{n}^2)$ (each from $O(\boldsymbol{n})$ literals has the size $O(\boldsymbol{n})$), and the maximal number of premises in the rules is $\boldsymbol{k} \leq 2$. After substituting these parameters to (3.24), we obtain:

$$\boxed{\boldsymbol{t} \ \leq \ 2^{O(\boldsymbol{n})}} \tag{C.6}$$

However, we should remember that our saturation procedure is non-deterministic because of the Splitting rule.

C.1.3 Complexity of the Procedure for $\mathcal{M}_f$

In this we estimate complexity of our decision procedure for the *full monadic fragment* given in subsection 4.1.3, Table 4.9.

Space complexity

The estimation for the number of clauses from (**T**) is relatively easy to carry out:

The number of clauses of form 1: If we fix a functional term $f(\overline{x})$ with $|\overline{x}| \leq 1$, then the number of literals of form $l[f(\overline{x}), \overline{x}]$ that can be obtained in saturation is bounded by by $\boldsymbol{L}_{1,f(\overline{x})} \leq O(\boldsymbol{n})$ (recall these literals must be instances of some initial literals), and so, the number of clauses for these parameters is bounded by $\boldsymbol{c}_{1,f(\overline{x})} = 2^{\boldsymbol{L}_{1,f(\overline{x})}} \leq 2^{O(\boldsymbol{n})}$. Summing up these values for all $f(\overline{x})$ with $|\overline{x}| \leq 1$, we obtain: $\boxed{\boldsymbol{c}_1 \leq O(\boldsymbol{n}) \cdot \boldsymbol{c}_{1,f(\overline{x})} \leq 2^{O(\boldsymbol{n})}}$.

The number of clauses of form 2: Let us fix a vector of variables $(\overline{x}, y)$ used in clauses of this form. Its length is at most $\boldsymbol{n}$: the length of such a vector is bounded by the maximal arity of a definitional predicate, which is $\boldsymbol{w} \leq \boldsymbol{n}$. For every choice of such vector, we have at most $O(\boldsymbol{n})$ literals of forms $l_m(\overline{x}, y)$ and $l_m(\overline{x})$ and at most $O(\boldsymbol{n}^2)$ unary literals of form $l^1_m[\overline{x}, y]$ (we have additionally $\boldsymbol{n}$ choices to pick a variable for every literal symbol). Hence the number of the clauses that can be constructed from these literals is bounded by $\boldsymbol{c}_{2,(\overline{x},y)} \leq 2^{O(\boldsymbol{n}^2)}$. Multiplying this estimation to the number of different variable-vectors of form $(\overline{x}, y)$ of length $\boldsymbol{n}$, we obtain: $\boxed{\boldsymbol{c}_2 \leq n^n \cdot \boldsymbol{c}_{2,(\overline{x},y)} \leq 2^{O(\boldsymbol{n}^2)}}$.

The number of clauses of form 3: Every clause of form 3 can be obtained from some clause of form 2 by a substitution of functional term $f(\overline{x})$ for variable y. Hence the number of such clauses is bounded by: $\boxed{\boldsymbol{c}_3 \leq \boldsymbol{f} \cdot \boldsymbol{c}_2 \leq 2^{O(\boldsymbol{n}^2)}}$.

The number of clauses of form 4: The estimation is analogous to the previous cases: $\boxed{\boldsymbol{c}_4 \leq 2^{O(\boldsymbol{n}^2)}}$.

So the maximal number of clauses that can be produced for a monadic formula is bounded by:

$$\boxed{\boldsymbol{c} \;=\; \boldsymbol{c}_1 + \boldsymbol{c}_2 + \boldsymbol{c}_3 + \boldsymbol{c}_4 \;\leq\; \boxed{2^{O(\boldsymbol{n}^2)}}} \tag{C.7}$$

A better bound $\boxed{\boldsymbol{c} \;\leq 2^{O(\boldsymbol{n})}}$ can be extracted after the following observations:

Observation 1: *Note that for initial clauses of form* 2, 3 *and* 4, *variable vectors* $(\overline{x})$ *and* $(\overline{x}, y)$ *consist of disjoint variables and this property is preserved under inferences given in Table 4.9. Hence the number of such variable-vectors (up to renaming) is* $O(\boldsymbol{n})$.

Observation 2: *The maximal number of unary literals of form* $l^1[\overline{x}, y]$, $l^1[\overline{x}, f(\overline{x})]$ *and* $l^1[\overline{x}, y]$ *that may occur in every clause of form* 2, 3 *and* 4 *is at most* $O(\boldsymbol{n})$ *either: every literal symbol correspond uniquely to the position of its variable in the vector of variables of a clause, and this correspondence does not change after inferences.*

C.2 Paramodulation-Based Decision Procedures

C.2.1 Complexity of the Procedure for $\mathcal{GF}_{\simeq}$

In order to estimate the complexity of a decision procedure given in Table 4.13, we compute, as usual, the maximal number of relevant clauses that can be generated for a guarded formula with equality. Unfortunately, refined calculations given for the clause class (**G**) in subsection 4.1.1 on p. 238, do not extend directly to clause class ($\mathbf{G}^{\simeq}$), because Observation 1 does not hold for paramodulation inferences anymore. Indeed, if we apply Ordered Paramodulation, say, for atoms $\underline{f(x)} \simeq g(x)$ and $\underline{a(\underline{f(x)}, x)}$, we obtain a new atom $a(g(x), x)$ that is *not* an instance of any of these atoms. Fortunately, we can find a weaker invariant, that suffices to extract the same complexity bounds:

Observation 1′: *All literals in a conclusion of* $\mathcal{OP}^{\succ}_{Sel}$*-inferences are obtained from literals in its premises, by replacing some terms with other ones. In addition, if the Ordered Paramodulation rule is applied simultaneously, then different occurrences of the same term would be replaced with the same terms. Therefore, all literals that may appear in saturation are obtained from some initial literals by such a replacement.*

Observations 2 and 3 given for the clauses from ($\mathbf{G}$) can be straightforwardly extended for class ($\mathbf{G}^{\simeq}$) and paramodulation inferences. Applying similar calculations, we obtain the same space and time complexity bounds of our paramodulation-based procedure:

$$\boxed{\boldsymbol{c},\, \boldsymbol{t} \;\le\; 2^{\boldsymbol{n}\cdot 2^{\boldsymbol{w}\cdot(\log \boldsymbol{w}+\epsilon)}}} \tag{C.8}$$

C.2.2 Complexity of the Procedure for $\mathcal{GF}_{\simeq}$ with Constants

In this section we estimate the complexity of two procedures for the guarded fragment with equality and constants, given, respectively in Table 4.15 and in Table 4.17 of subsection 4.3.2.

The complexity of the procedure through elimination of constants

Let us estimate the maximal number of clauses from class ($\mathbf{G}^{\overline{z}}$) given in Table 4.14. For a fixed prefix $\boldsymbol{v}$ of constants (of the length $\boldsymbol{k}$), and a fixed group Fun_i of other (Skolem) constants (of the size at most $\boldsymbol{w}$), we may obtain at most $O(\boldsymbol{n}\cdot(\boldsymbol{w}+\boldsymbol{k})^{\boldsymbol{w}})$ literals of form $l(\boldsymbol{v},\{\hat{c}\})$ or $(\simeq|\not\simeq)[\hat{c}]$ from the initial literals by simulteneous replacement of its arguments: initial $\boldsymbol{k}$ arguments we must replace with constants to match $\boldsymbol{v}$ (we do not have a choice for this), for the remaining at most $\boldsymbol{w}$ *other* arguments, we have a choice between constants from $\boldsymbol{v}$, or constants from Fun_i. This gives us at most $2^{O(\boldsymbol{n}\cdot(\boldsymbol{w}+\boldsymbol{k})^{\boldsymbol{w}})}$ clauses of type 1, which we should multiply on $\boldsymbol{k}^{\boldsymbol{k}}$ choices for $\boldsymbol{v}$ and $\boldsymbol{n}$ choices for Fun_i. The number of clauses of type 2 is estimated similarly (here we also fix a vector of variables $\overline{x}$ with at most $\boldsymbol{w}$ variables which do not belong to $\overline{z}$) to be bounded by $\boldsymbol{k}^{\boldsymbol{k}}\cdot(\boldsymbol{w}+\boldsymbol{k})^{(\boldsymbol{w}+\boldsymbol{k})}\cdot\boldsymbol{n}\cdot 2^{O(\boldsymbol{n}\cdot(2\boldsymbol{w}+\boldsymbol{k})^{\boldsymbol{w}})} \le 2^{\boldsymbol{n}\cdot(2\boldsymbol{w}+\boldsymbol{k})^{(\boldsymbol{w}+\epsilon)}}$ Hence the maximal number of clauses from ($\mathbf{G}^{\overline{z}}$), and the time required to compute the saturation are bounded by:

$$\boxed{\boldsymbol{c},\; \boldsymbol{t} \;\le\; 2^{\boldsymbol{n}\cdot(2\boldsymbol{w}+\boldsymbol{k})^{(\boldsymbol{w}+\epsilon)}}} \tag{C.9}$$

which is now optimal even for $\mathcal{GF}^{k}_{\simeq}$ with constants (since $\boldsymbol{k}\le\boldsymbol{n}$).

The complexity of the direct procedure

In order to estimate the complexity described in Table 4.16 , note, that for every clause that is derived according to our strategy, every literal is obtained from some initial literals by (*i*) *simulteneously* replacing non-constant arguments (their number is at most $\boldsymbol{w}$) with some terms and (*ii*) applying a substitution of constants for constants, which is *fixed* for a clause (this substitution is *uniquely determined* by unit clauses of form 1.1.4). The number of such clauses can be computed analogously as in non-direct approach (see the calculations before). Although the procedure is non-deterministic, we may notice that every branch is uniquely determined by a set of (in)equalities between constants obtained after splitting clauses of form 1.0. Hence the number of such branches is at most $O(2^{\boldsymbol{k}^2})$.[1] Multiplying the number of clauses on the number of branches, we obtain the same complexity as in (C.9).

[1] This is a very rough, but still sufficient estimation. In fact, the number of branches is bounded by the number of partitions of constants on equivalent classes, which is at most $\boldsymbol{k}^{\boldsymbol{k}}$

Appendix D

$\mathcal{GF}$ with Compositional Guards

In this appendix we give some technical details on decision procedures for extensions of the guarded fragment with compositional axioms described in chapter 5.

D.1 Redundancy Lemmas

In this appendix we give the proofs for the lemmas showing redundancy of inferences with compositional relations formulated in chapter 5.

Proof of Lemma 5.2.6. According to the definition of redundancy for inferences, it suffices to prove the lemma only for the case when clause 1 is ground:

Inference ($\boldsymbol{a}$): For any ground instance of the first inference:

1. $C \vee \boldsymbol{\underline{s}Tt}$;
$\mathtt{R}'_0$. $\boldsymbol{\neg(\underline{s}Th)} \vee \alpha[s] \vee \beta[h]$;
$\mathtt{NC}[1_a; \mathtt{R}'_0]$: $3'_a$. $C \vee \neg(tTh) \vee \alpha[s] \vee \beta[h]$

there should be $s \succ t$, $s \succ h$ and $sTt \succ C$ because of, respectively, the conditions (iv), (v) and (ii) of the Negative Chaining rule (see System 6 on p.84). The conclusion of this inference follows from the clause 2_a and the following instances of the clauses $\mathtt{R}_2$ and $\mathtt{R}_3$:

2_a. $C \vee \alpha[s] \vee \underline{u(t)}$.
$\mathtt{R}'_2$. $\boldsymbol{\neg(tTh)} \vee \neg\underline{u(t)} \vee \underline{u(h)}$;
$\mathtt{R}'_3$. $\neg\underline{u(h)} \vee \beta[h]$;

All the above clauses are smaller than the clause $\mathtt{R}'_0$ which has been used in the inference. This can be shown using the following sequences of ordering relations between literals:

$(\mathbf{1})\ \neg(sTh) \succ sTt \succ C;$
$(\mathbf{2})\ \neg(sTh) \succ sTh \succ \neg(tTh) \succ \neg u(t) \succ u(t);$
$(\mathbf{3})\ \neg(sTh) \succ \neg u(h) \succ u(h).$

The first sequence follows from the condition (C1) of admissible orderings (see Definition 3.5.4 on p.84) and condition (i) of the Negative Chaining rule. The second sequence is follows from conditions (R1) (see Definition 3.5.1 on p.79) and (C3) of admissible ordering, and the fact that $\succ$ is *CASP* (see Definition 5.2.5), which also imply the third sequence. Now $\mathtt{R}_0' \succ 2_a$ follows from (1) and (2), $\mathtt{R}_0' \succ \mathtt{R}_2'$ follows from (2) and (3), and $\mathtt{R}_0' \succ \mathtt{R}_3'$ follows from (3). Since the result of the inference follows from smaller clauses than its maximal premise, the inference is redundant.

<u>Inference ($\boldsymbol{b}$)</u>: Similarly to case (a), for any ground instance of the second inference:

1. $C \vee \boldsymbol{tT\underline{s}};$
$\mathtt{R}_0''.\ \boldsymbol{\neg(hT\underline{s})} \vee \alpha[h] \vee \beta[s];$
$\mathtt{NC}[1;\mathtt{R}_0'']\text{: } 3_b'.\quad C \vee \neg(hTt) \vee \alpha[h] \vee \beta[s]$

there should be $s \succ t$, $s \succeq h$ and $tTs \succ C$ because of the conditions of the Negative Chaining rule. The conclusion $3_b'$ of the inference is a logical consequence of the following instances of the clauses $\mathtt{R}_1$, 2_b and $\mathtt{R}_3$:

$\mathtt{R}_1'.\ \boldsymbol{\neg(hTt)} \vee \alpha[h] \vee \underline{u(t)};$
$2_b.\ C \vee \neg \underline{u(t)} \vee \underline{u(s)};$
$\mathtt{R}_3''.\ \neg \underline{u(s)} \vee \beta[s];$

Again, the following sequences of ordering relations between literals can be derived:

$(\mathbf{1})\quad \neg(hTs) \succ tTs \succ C,\quad \text{when } s \succ h;$
$(\mathbf{1a})\ \neg(sTs) \succ tTs \succ C;$
$(\mathbf{2})\quad \neg(hTs) \succ hTs \succ \neg(hTt) \succ \neg u(t) \succ u(t),\quad \text{when } s \succ h;$
$(\mathbf{2a})\ \neg(sTs) \succ \neg(sTt) \succ \neg u(t) \succ u(t);$
$(\mathbf{3})\quad \neg(hTs) \succ \neg u(s) \succ u(s).$

where (1), (2), (3) are symmetric analog of those from case (a), and (1a), (2a) cover a special case $s = h$, which is possible for Negative Chaining with right arguments. For them we should use the last cases from conditions (C1) and (C2) of admissible ordering. Now $\mathtt{R}_0'' \succ \mathtt{R}_1'$ follows from (2) and (2a), $\mathtt{R}_0'' \succ 2_b$ follows from (1), (1a), (2), (2a) and (3), and $\mathtt{R}_0'' \succ \mathtt{R}_3''$ follows from (3). This completes the proof of redundancy for inference (b). ⧉

Proof of <u>Lemma 5.2.9</u>. This lemma is proven analogously to Lemma 5.2.6. As usual, we may assume that the clauses 1_a and 1_b are ground.

<u>Case ($\boldsymbol{a}$)</u>: For any ground instance of the first inference:

1_a. $C \vee \boldsymbol{\underline{s}St}$;
$\mathtt{R}'_0$. $\boldsymbol{\neg(\underline{s}Hh)} \vee \alpha[s] \vee \beta[h]$;
$\mathtt{NC}[1_a;\mathtt{R}'_0]$: $3'_a$. $C \vee \neg(tTh) \vee \alpha[s] \vee \beta[h]$

we have $s \succ t$, $s \succ h$ and $sSt \succ C$. The conclusion of the inference follows from clause 2_a and the following instances of clauses $\mathtt{R}_2$ and $\mathtt{R}_3$:

2_a. $C \vee \alpha[s] \vee \underline{u_1(t)}$.
$\mathtt{R}'_2$. $\boldsymbol{\neg(tTh)} \vee \underline{\neg u_1(t)} \vee \underline{u_2(h)}$;
$\mathtt{R}'_3$. $\underline{\neg u_2(h)} \vee \beta[h]$;

All the above clauses are $\succ$-smaller than the clause $\mathtt{R}'_0$ which has been used in the inference because of the following sequences of inequalities:

(**1**) $\neg(sHh) \succ sSt \succ C$	[(C1), $\mathtt{NC}.(i)$];
(**2**) $\neg(sHh) \succ sHh \succ \neg(tTh) \succ \neg u_1(t) \succ u_1(t)$	[(R1), (C3), *CASP*];
(**3**) $\neg(sHh) \succ \neg u_2(h) \succ u_2(h)$	[*CASP*, (R1)];

Therefore, the inference producing the clause $3'_a$ is redundant.

Case (***b***): For any ground instance of the second inference:

1. $C \vee \boldsymbol{tT\underline{s}}$;
$\mathtt{R}''_0$. $\boldsymbol{\neg(hH\underline{s})} \vee \alpha[h] \vee \beta[s]$;
$\mathtt{NC}[1;\mathtt{R}''_0]$: $3'_b$. $C \vee \neg(hSt) \vee \alpha[h] \vee \beta[s]$

there should be $s \succ t$, $s \succeq h$ and $tTs \succ C$. The conclusion $3'_b$ of the inference follows from the following instances of the clauses $\mathtt{R}_1$, 2_b and $\mathtt{R}_3$:

$\mathtt{R}'_1$. $\boldsymbol{\neg(hSt)} \vee \alpha[h] \vee \underline{u_1(t)}$;
2_b. $C \vee \underline{\neg u_1(t)} \vee \underline{u_2(s)}$;
$\mathtt{R}''_3$. $\underline{\neg u_2(s)} \vee \beta[s]$;

All the above clauses are $\succ$-smaller than the clause $\mathtt{R}''_0$, because of the following sequences of inequalities (in (1) and (2) we assume that $s \succ h$):

(**1**) $\neg(hHs) \succ tTs \succ C$,	[(C1), $\mathtt{NC}.(i)$];
(**1a**) $\neg(sHs) \succ tTs \succ C$	[(C1), $\mathtt{NC}.(i)$];
(**2**) $\neg(hHs) \succ hHs \succ \neg(hSt) \succ \neg u_1(t) \succ u_1(t)$	[(R1), (C3), *CASP*];
(**2a**) $\neg(sHs) \succ \neg(sSt) \succ \neg u_1(t) \succ u_1(t)$	[(C2), *CASP*];
(**3**) $\neg(hHs) \succ \neg u_2(s) \succ u_2(s)$	[*CASP*, (R1)];

This completes the proof of redundancy for the inference in case (b). ◫

Proof of Lemma 5.2.10. If a Negative Chaining between clauses 1 and 2 is possible, then $S' \circ T' \subseteq H' = U$ for some S', T' and H', and either (***a***) $R = S'$ (for left chaining), or (***b***) $R = T'$ (for right chaining). By condition of this lemma, N should contain the following clauses:

$\mathtt{TC}[2]$: $3'$. $\boldsymbol{\neg(xS'y)} \vee \alpha[x] \vee u_\alpha^{S'}(y)$; $\mathtt{OR}[1;3']$: $6'$. $C \vee \alpha[s] \vee u_\alpha^{S'}(t)$ *if* $R = S'$;
$4'$. $\boldsymbol{\neg(xT'y)} \vee \neg u_\alpha^{S'}(x) \vee u_\alpha^{H'}(y)$; $\mathtt{OR}[1;4']$: $7'$. $C \vee \neg u_\alpha^{S'}(s) \vee u_\alpha^{H'}(t)$ *if* $R = T'$;
$5'$. $\neg u_\alpha^{H'}(y) \vee \beta[y]$;

Now, redundancy follows from Lemma 5.2.9 with $\mathtt{R}_0 := 2$, $\mathtt{R}_1 := 3'$, $\mathtt{R}_3 := 4'$ and $\mathtt{R}_4 := 5'$.

Redundancy of Negative Chaining inferences with clause 3 follows similarly as above: except that now $S' \circ T' \subseteq H' = S$ and $\beta[y]$ in clause $5'$ must be replaced with $\neg u_\alpha^{S}[y] = u_\alpha^{H'}[y]$, which makes clause $5'$ redundant w.r.t. N.

To prove redundancy of Negative Chaining inferences with clause 4, i.e., when $S' \circ T' \subseteq H' = T$ and either (***a***) $R = S'$, or (***b***) $R = T'$, note than since $S \circ (S' \circ T') \subseteq H$, by *associativity of composition* (see subsection 3.5.4, p.82), there must be some S'' such that $S \circ S' \subseteq S''$ and $S'' \circ T' \subseteq H$. Hence N contains the following clauses:

$\mathtt{TC}[2]$: $4'$. $\boldsymbol{\neg(xS'y)} \vee \neg u_\alpha^{S}(x) \vee u_\alpha^{S''}(y)$; $\mathtt{OR}[1;4']$: $6'$. $C \vee \alpha[s] \vee u_\alpha^{S}(t)$ *if* $R = S'$;
$4''$. $\boldsymbol{\neg(xT'y)} \vee \neg u_\alpha^{S''}(x) \vee u_\alpha^{H}(y)$; $\mathtt{OR}[1;4'']$: $7'$. $C \vee \neg u_\alpha^{S}(s) \vee u_\alpha^{H}(t)$ *if* $R = T'$;

Now, redundancy of $[1;4]$ follows from Lemma 5.2.9 with $\mathtt{R}_0 := 4$, $\mathtt{R}_1 := 4'$, $\mathtt{R}_3 := 4''$ and $\mathtt{R}_4 := \neg u_\alpha^{H}(y) \vee u_\alpha^{H}[y]$. ⧉

D.2 Deciding The Guarded Fragment With Transitive Guards

Table D.1: Possible inferences between clauses for the guarded fragment with transitive guards

$a := p \mid T;\quad l := p \mid \neg p \mid \neg T;\quad \alpha := \vee\{l\};$
$q := p \mid u^T_{\hat{\alpha}};\quad b := a \mid u^T_{\hat{\alpha}};\quad k := l \mid u^T_{\hat{\alpha}} \mid \neg u^T_{\hat{\alpha}};\quad \beta := \vee\{k\};$

1	$\hat{\beta}[\hat{c}]$	
1.1	$\hat{\beta}[\hat{c}] \vee \boldsymbol{k[\hat{c}]}$	
1.1.1	$\hat{\beta}[\hat{c}] \vee \underline{\boldsymbol{q[\hat{c}]}}^{\star}$	$:\mathtt{OR.1}$
1.1.2	$\hat{\beta}[\hat{c}] \vee \neg\underline{\boldsymbol{q[\hat{c}]}}$	$:\mathtt{OR.2}$
1.1.3	$\hat{\beta}[\hat{c}] \vee \neg\underline{\boldsymbol{T[\hat{c}]}}$	$:\mathtt{OR.2}$
1.1.4	$\hat{\beta}[\hat{c}] \vee \neg\boldsymbol{(\underline{c_1}\tilde{T}c_2)}$	$:\mathtt{NC}$
	$\mathtt{OR}[1.1.1; 1.1.2]\colon \hat{\beta}[\hat{c}]$	$:1$
2	$\neg\hat{a}[!\overline{x}] \vee \hat{\alpha}[\overline{x}] \vee \hat{\beta}[!\hat{f}(\overline{x}), \overline{x}]$	
2.1	$\neg\hat{a}[!\overline{x}] \vee \hat{\alpha}[\overline{x}] \vee \hat{\beta}[!\hat{f}(\overline{x}), \overline{x}] \vee \boldsymbol{k[!\hat{f}(\overline{x}), \overline{x}]}$	
2.1.1	$\neg\hat{a}[!\overline{x}] \vee \hat{\alpha}[\overline{x}] \vee \hat{\beta}[!\hat{f}(\overline{x}), \overline{x}] \vee \underline{\boldsymbol{q[!\hat{f}(\overline{x}), \overline{x}]}}^{\star}$	$:\mathtt{OR.1}$
2.1.2	$\neg\hat{a}[!\overline{x}] \vee \hat{\alpha}[\overline{x}] \vee \hat{\beta}[!\hat{f}(\overline{x}), \overline{x}] \vee \neg\underline{\boldsymbol{q[!\hat{f}(\overline{x}), \overline{x}]}}$	$:\mathtt{OR.2}$
2.1.3	$\neg\hat{a}[!\overline{x}] \vee \hat{\alpha}[\overline{x}] \vee \hat{\beta}[!\hat{f}(\overline{x}), \overline{x}] \vee \underline{q[\hat{f}(\overline{x}), \overline{x}]} \vee \underline{\boldsymbol{q[!\hat{f}(\overline{x}), \overline{x}]}}$	$:\mathtt{OF}$
2.1.4	$\neg\hat{a}[!\overline{x}] \vee \hat{\alpha}[\overline{x}] \vee \hat{\beta}[!\hat{f}(\overline{x}), \overline{x}] \vee \neg\underline{\boldsymbol{T[!\hat{f}(\overline{x}), \overline{x}]}}$	$:\mathtt{OR.2}$
2.1.5	$\neg\hat{a}[!\overline{x}] \vee \hat{\alpha}[\overline{x}] \vee \hat{\beta}[!\hat{f}(\overline{x}), \overline{x}] \vee \neg\boldsymbol{(\underline{f(\overline{x})}\tilde{T}(\hat{f}(\overline{x}) \mid \overline{x}))}$	$:\mathtt{NC}$
	$\mathtt{OR}[2.1.1; 2.1.2]\colon \neg\hat{a}[!\overline{x}] \vee \hat{\alpha}[\overline{x}] \vee \hat{\beta}[!\hat{f}(\overline{x}), \overline{x}]$	$:2$
	$\mathtt{OF}[2.1.3]\colon \neg\hat{a}[!\overline{x}] \vee \hat{\alpha}[\overline{x}] \vee \hat{\beta}[!\hat{f}(\overline{x}), \overline{x}] \vee q[!\hat{f}(\overline{x}), \overline{x}]$	$:2$
2.2	$\neg\boldsymbol{\hat{q}[!\overline{x}]}^{\sharp} \vee \hat{\alpha}[\overline{x}]$	
2.2.1	$\neg\underline{\boldsymbol{q[!\overline{x}]}} \vee \hat{\alpha}[\overline{x}]$	$:\mathtt{OR.2}$
	$\mathtt{OR}[1.1.1; 2.2.1]\colon \hat{\beta}[\hat{c}] \vee \hat{\alpha}[\hat{c}]$	$:1$
	$\mathtt{OR}[2.1.1; 2.2.1]\colon \neg\hat{a}[!\overline{x}] \vee \hat{\alpha}[\overline{x}] \vee \hat{\beta}[!\hat{f}(\overline{x}), \overline{x}]$	$:2$
2.3	$\neg\hat{T}[!\overline{x}] \vee \hat{\alpha}[\overline{x}] \vee \boldsymbol{p[!\overline{x}]}$	
2.3.1	$\neg\hat{T}[!\overline{x}] \vee \hat{\alpha}[\overline{x}] \vee \underline{\boldsymbol{p[!\overline{x}]}}^{\star}$	$:\mathtt{OR.1}$
2.3.2	$\neg\hat{T}[!\overline{x}] \vee \hat{\alpha}[\overline{x}] \vee \underline{p[\overline{x}]} \vee \underline{\boldsymbol{p[!\overline{x}]}}$	$:\mathtt{OF}$
	$\mathtt{OR}[2.3.1; 1.1.2]\colon \neg\hat{T}[\hat{c}] \vee \hat{\alpha}[\hat{c}] \vee \hat{\beta}[\hat{c}]$	$:1$
	$\mathtt{OR}[2.3.1; 2.1.2]\colon \neg\hat{T}[!\overline{x}] \vee \hat{\alpha}[\overline{x}] \vee \neg\hat{a}[!\overline{x}] \vee \hat{\alpha}[\overline{x}] \vee \hat{\beta}[!\hat{f}(\overline{x}), \overline{x}]$	$:2$
	$\mathtt{OR}[2.3.1; 2.2.1]\colon \neg\hat{T}[!\overline{x}] \vee \hat{\alpha}[\overline{x}] \vee \hat{\alpha}[\overline{x}]$	$:3$
	$\mathtt{OF}[2.3.2]\colon \neg\hat{T}[!\overline{x}] \vee \hat{\alpha}[\overline{x}] \vee p[!\overline{x}]$	$:3$
2.4	$\llbracket \neg\hat{T}[!x, !y] \vee \hat{\alpha}[x, y] \vee \boldsymbol{l[!x]}^{\sharp} \rrbracket$	$:\mathtt{LP}$
	$\mathtt{LP}[2.4]\colon \neg\hat{T}[!x, !y] \vee \hat{\alpha}[x, y] \vee p_{l[\cdot]}(x)$	$:2$
	$\colon \neg p_{l[\cdot]}(x) \vee l[!x]$	$:2$

Continued on next page

2.5	$\neg\boldsymbol{T}[!\overline{\boldsymbol{x}}] \vee \neg\hat{T}[!\overline{x}] \vee \hat{\alpha}[\overline{x}]$	
2.5.1	$\neg\hat{T}[!x] \vee \hat{\alpha}[x]$	:`U`
2.5.2	$\neg\{!\hat{T}\}[!x, !y] \vee \hat{\alpha}[x] \vee \hat{\alpha}[y]$	:`T`
`T.`	$\neg\boldsymbol{\{!\hat{T}\}[!x, !y]}^{\sharp} \vee \hat{\alpha}[x] \vee \hat{\alpha}[y]$	
`T.1`	$\neg\underline{\boldsymbol{(xTy)}} \vee \hat{\beta}[x, y]$	:`OR.2`
`T.2`	$\neg\boldsymbol{(xTy)}^{\sharp} \vee \hat{\alpha}[x] \vee \hat{\alpha}[y]$	:`TC`
`T.3`	$\neg\boldsymbol{(\underline{x}Ty)} \vee \hat{\alpha}[x] \vee \hat{\alpha}[y]$	:$\llbracket$`NC.l`$\rrbracket$: Lemma 5.2.7
`T.4`	$\neg\boldsymbol{(xT\underline{y})} \vee \hat{\alpha}[x] \vee \hat{\alpha}[y]$	:$\llbracket$`NC.r`$\rrbracket$: Lemma 5.2.7
`T.5`	$\neg\boldsymbol{(\underline{x}\tilde{T}_1 y)}^{\sharp} \vee \cdots \vee \neg\boldsymbol{(\underline{x}\tilde{T}_n y)}^{\sharp} \vee \hat{\alpha}[x] \vee \hat{\alpha}[y],\ n \geq 2$	:`HC`
`T.6`	$\neg\boldsymbol{(\underline{x}\tilde{T}_1 y)}^{\sharp} \vee \cdots \vee \neg\boldsymbol{(\underline{y}\tilde{T}_n x)}^{\sharp} \vee \hat{\alpha}[x] \vee \hat{\alpha}[y],\ n \geq 2$	:`HC`
`TC`[`T.2`]: `T.2.1`	$\neg\boldsymbol{(xTy)}^{\sharp} \vee \hat{\alpha}[x] \vee u^{T}_{\hat{\alpha}}(y)$	
`T.2.1.1`	$\neg\underline{\boldsymbol{(xTy)}} \vee \hat{\alpha}[x] \vee u^{T}_{\hat{\alpha}}(y)$	:`OR.2` ⇒ `T.1`
`T.2.1.2`	$\neg\boldsymbol{(\underline{x}Ty)} \vee \hat{\alpha}[x] \vee u^{T}_{\hat{\alpha}}(y)$	:$\llbracket$`NC.l`$\rrbracket$: Lemma 5.2.7
`T.2.1.3`	$\neg\boldsymbol{(xT\underline{y})} \vee \hat{\alpha}[x] \vee u^{T}_{\hat{\alpha}}(y)$	:$\llbracket$`NC.r`$\rrbracket$: Lemma 5.2.7
`T.2.2`	$\neg\boldsymbol{(xTy)}^{\sharp} \vee \neg u^{T}_{\hat{\alpha}}(x) \vee u^{T}_{\hat{\alpha}}(y)$	
`T.2.2.1`	$\neg\underline{\boldsymbol{(xTy)}} \vee \neg u^{T}_{\hat{\alpha}}(x) \vee u^{T}_{\hat{\alpha}}(y)$	:`OR.2` ⇒ `T.1`
`T.2.2.2`	$\neg\boldsymbol{(\underline{x}Ty)} \vee \neg u^{T}_{\hat{\alpha}}(x) \vee u^{T}_{\hat{\alpha}}(y)$	:$\llbracket$`NC.l`$\rrbracket$: Lemma 5.2.7
`T.2.2.3`	$\neg\boldsymbol{(xT\underline{y})} \vee \neg u^{T}_{\hat{\alpha}}(x) \vee u^{T}_{\hat{\alpha}}(y)$	:$\llbracket$`NC.r`$\rrbracket$: Lemma 5.2.7
`T.2.3`	$\neg u^{T}_{\hat{\alpha}}(y) \vee \hat{\alpha}[y]$	:`U.4`
`U.1`	$\neg p(x) \vee \boldsymbol{h^{T}_{l}(x)Tx}$	
`U.1.1`	$\neg p(x) \vee \underline{\boldsymbol{T[h^{T}(x), x]}}^{\star}$	:`OR.1`
`U.1.2`	$\neg p(x) \vee \underline{\boldsymbol{h^{T}_{l}(x)}}\boldsymbol{Tx}^{\star}$	:`C.l`
`OR`[`U.1.1`; `T.1`]:	$\neg p(x) \vee \hat{\beta}[h^{T}(x), x]$	:`U.4`
`U.2`	$\neg p(x) \vee \boldsymbol{xTh^{T}_{r}(x)}$	
`U.2.1`	$\neg p(x) \vee \underline{\boldsymbol{xTh^{T}_{r}(x)}}^{\star}$	:`OR.1` ⇒ `U.1.1`
`U.2.2`	$\neg p(x) \vee \boldsymbol{xT}\underline{\boldsymbol{h^{T}_{r}(x)}}^{\star}$	:`C.r`
`HC`[{`U.2.2`}; `T.6`]:	$\neg\{\hat{T}\}[h(x), x] \vee \alpha[h(x), x]$	:`U.4`
`U.3`	$\neg p(x) \vee \boldsymbol{xTx}$	
`U.3.1`	$\neg p(x) \vee \underline{\boldsymbol{xTx}}^{\star}$	:`OR.1` ⇒ `U.1.1`

Continued on next page

U.4	$\beta[h^T(x), x]$	
U.4.1	$\beta[h^T(x), x] \vee \underline{\boldsymbol{q[!h^T(x), x]}}^{\star}$	:OR.1
U.4.2	$\beta[h^T(x), x] \vee \neg\underline{\boldsymbol{q[!h^T(x), x]}}$	:OR.2
U.4.3	$\beta[h^T(x), x] \vee \neg\underline{\boldsymbol{T[!h^T(x), x]}}$	:OR.2
U.4.4	$\beta[x] \vee \underline{\boldsymbol{q[!x]}}^{\star}$	:OR.1
U.4.5	$\beta[x] \vee \neg\underline{\boldsymbol{q[!x]}}$	:OR.2
U.4.6	$\beta[x] \vee \neg\underline{\boldsymbol{T[!x]}}$	:OR.2
U.4.7	$\beta[]$ $\Rightarrow$	:1
OR[U.4.1; 2.2.1]:	$\beta[h^T(x), x] \vee \hat{\alpha}[h^T(x), x]$	:U.4
OR[2.3.1; U.4.2]:	$\neg\hat{T}[!h^T(x), x] \vee \hat{\alpha}[h^T(x), x] \vee \beta[h^T(x), x]$	:U.4
OR[U.4.1; U.4.2]:	$\beta[h^T(x), x]$	:U.4
OR[U.1.1; U.4.3]:	$\neg p(x) \vee \beta[h^T(x), x]$	:U.4
OR[U.4.4; 1.1.2]:	$\beta[c] \vee \hat{\beta}[\hat{c}]$	:1
OR[1.1.1; U.4.5]:	$\hat{\beta}[\hat{c}] \vee \beta[c]$	:1
OR[U.4.4; 2.1.2]:	$\beta[!\hat{f}(\overline{x}), \overline{x}] \vee \neg\hat{a}[!\overline{x}] \vee \hat{\alpha}[\overline{x}] \vee \hat{\beta}[!\hat{f}(\overline{x}), \overline{x}]$	:2
OR[2.1.3; U.4.5]:	$\neg\hat{a}[!\overline{x}] \vee \hat{\alpha}[\overline{x}] \vee \hat{\beta}[!\hat{f}(\overline{x}), \overline{x}] \vee \beta[!\hat{f}(\overline{x}), \overline{x}]$	:2
OR[U.4.4; 2.2.1]:	$\beta[x] \vee \hat{\alpha}[x]$	:U.4
OR[2.3.1; U.4.5]:	$\neg\hat{T}[!x] \vee \hat{\alpha}[x] \vee \beta[x]$	:U.4
OR[U.4.4; U.4.5]:	$\beta[x]$	:U.4
OR[U.1.1; U.4.6]:	$\neg p(x) \vee \beta[x]$	:U.4

D.3 Deciding The Guarded Fragment With Compositional Guards

Table D.2: Possible inferences between clauses for the guarded fragment with compositional guards

$a := p \mid S;\quad l := p \mid \neg p \mid \neg S;\quad \alpha := \vee\{l\};$
$q := p \mid u^{S}_{\hat\alpha};\quad b := a \mid u^{S}_{\hat\alpha};\quad k := l \mid u^{S}_{\hat\alpha} \mid \neg u^{S}_{\hat\alpha};\quad \beta := \vee\{k\};$

1	$\hat\beta[\hat c]$	
1.1	$\hat\beta[\hat c] \vee \boldsymbol{k[\hat c]}$	
1.1.1	$\hat\beta[\hat c] \vee \underline{\boldsymbol{q[\hat c]}}^{\star}$	$:\texttt{OR.1}$
1.1.2	$\hat\beta[\hat c] \vee \neg\underline{\boldsymbol{q[\hat c]}}$	$:\texttt{OR.2}$
1.1.3	$\hat\beta[\hat c] \vee \neg\underline{\boldsymbol{S[\hat c]}}$	$:\texttt{OR.2}$
1.1.4	$\hat\beta[\hat c] \vee \neg\boldsymbol{(\underline{c_1}\tilde{S}c_2)}$	$:\texttt{NC}$
	$\texttt{OR}[1.1.1;1.1.2]\colon \hat\beta[\hat c]$	$:1$
2	$\neg\hat a[!\overline{x}] \vee \hat\alpha[\overline{x}] \vee \hat\beta[!\hat f(\overline{x}),\overline{x}]$	
2.1	$\neg\hat a[!\overline{x}] \vee \hat\alpha[\overline{x}] \vee \hat\beta[!\hat f(\overline{x}),\overline{x}] \vee \boldsymbol{k[!\hat f(\overline{x}),\overline{x}]}$	
2.1.1	$\neg\hat a[!\overline{x}] \vee \hat\alpha[\overline{x}] \vee \hat\beta[!\hat f(\overline{x}),\overline{x}] \vee \underline{\boldsymbol{q[!\hat f(\overline{x}),\overline{x}]}}^{\star}$	$:\texttt{OR.1}$
2.1.2	$\neg\hat a[!\overline{x}] \vee \hat\alpha[\overline{x}] \vee \hat\beta[!\hat f(\overline{x}),\overline{x}] \vee \neg\underline{\boldsymbol{q[!\hat f(\overline{x}),\overline{x}]}}$	$:\texttt{OR.2}$
2.1.3	$\neg\hat a[!\overline{x}] \vee \hat\alpha[\overline{x}] \vee \hat\beta[!\hat f(\overline{x}),\overline{x}] \vee \underline{q[\hat f(\overline{x}),\overline{x}]} \vee \underline{\boldsymbol{q[!\hat f(\overline{x}),\overline{x}]}}$	$:\texttt{OF}$
2.1.4	$\neg\hat a[!\overline{x}] \vee \hat\alpha[\overline{x}] \vee \hat\beta[!\hat f(\overline{x}),\overline{x}] \vee \neg\underline{\boldsymbol{S[!\hat f(\overline{x}),\overline{x}]}}$	$:\texttt{OR.2}$
2.1.5	$\neg\hat a[!\overline{x}] \vee \hat\alpha[\overline{x}] \vee \hat\beta[!\hat f(\overline{x}),\overline{x}] \vee \neg\boldsymbol{(\underline{f(\overline{x})}\tilde{S}(\hat f(\overline{x})\mid\overline{x}))}$	$:\texttt{NC}$
	$\texttt{OR}[2.1.1;2.1.2]\colon \neg\hat a[!\overline{x}] \vee \hat\alpha[\overline{x}] \vee \hat\beta[!\hat f(\overline{x}),\overline{x}]$	$:2$
	$\texttt{OF}[2.1.3]\colon \neg\hat a[!\overline{x}] \vee \hat\alpha[\overline{x}] \vee \hat\beta[!\hat f(\overline{x}),\overline{x}] \vee q[!\hat f(\overline{x}),\overline{x}]$	$:2$
2.2	$\neg\boldsymbol{\hat q[!\overline{x}]}^{\sharp} \vee \hat\alpha[\overline{x}]$	
2.2.1	$\neg\underline{\boldsymbol{p[!\overline{x}]}} \vee \hat\alpha[\overline{x}]$	$:\texttt{OR.2}$
	$\texttt{OR}[1.1.1;2.2.1]\colon \hat\beta[\hat c] \vee \hat\alpha[\hat c]$	$:1$
	$\texttt{OR}[2.1.1;2.2.1]\colon \neg\hat a[!\overline{x}] \vee \hat\alpha[\overline{x}] \vee \hat\beta[!\hat f(\overline{x}),\overline{x}]$	$:2$
2.3	$\neg\hat S[!\overline{x}] \vee \hat\alpha[\overline{x}] \vee \boldsymbol{p[!\overline{x}]}$	
2.3.1	$\neg\hat S[!\overline{x}] \vee \hat\alpha[\overline{x}] \vee \underline{\boldsymbol{p[!\overline{x}]}}^{\star}$	$:\texttt{OR.1}$
2.3.2	$\neg\hat S[!\overline{x}] \vee \hat\alpha[\overline{x}] \vee \underline{p[\overline{x}]} \vee \underline{\boldsymbol{p[!\overline{x}]}}$	$:\texttt{OF}$
	$\texttt{OR}[2.3.1;1.1.2]\colon \neg\hat S[\hat c] \vee \hat\alpha[\hat c] \vee \hat\beta[\hat c]$	$:1$
	$\texttt{OR}[2.3.1;2.1.2]\colon \neg\hat S[!\overline{x}] \vee \hat\alpha[\overline{x}] \vee \neg\hat a[!\overline{x}] \vee \hat\alpha[\overline{x}] \vee \hat\beta[!\hat f(\overline{x}),\overline{x}]$	$:2$
	$\texttt{OR}[2.3.1;2.2.1]\colon \neg\hat S[!\overline{x}] \vee \hat\alpha[\overline{x}] \vee \hat\alpha[\overline{x}]$	$:3$
	$\texttt{OF}[2.3.2]\colon \neg\hat S[!\overline{x}] \vee \hat\alpha[\overline{x}] \vee p[!\overline{x}]$	$:3$
2.4	$\llbracket \neg\hat S[!x,!y] \vee \hat\alpha[x,y] \vee \boldsymbol{l[!x]}^{\sharp} \rrbracket$	$:\texttt{LP}$
	$\texttt{LP}[2.4]\colon \neg\hat S[!x,!y] \vee \hat\alpha[x,y] \vee p_{l[\cdot]}(x)$	$:2$
	$\colon \neg p_{l[\cdot]}(x) \vee l[!x]$	$:2$

Continued on next page

2.5	$\neg\boldsymbol{S}[!\overline{\boldsymbol{x}}] \vee \neg\hat{S}[!\overline{x}] \vee \hat{\alpha}[\overline{x}]$	
2.5.1	$\neg\hat{S}[!x] \vee \hat{\alpha}[x]$	:U
2.5.2	$\neg\{!\hat{S}\}[!x, !y] \vee \hat{\alpha}[x] \vee \hat{\alpha}[y]$	:S

S.	$\neg\boldsymbol{\{!\hat{S}\}[!x, !y]}^{\sharp} \vee \hat{\alpha}[x] \vee \hat{\alpha}[y]$	
S.**1**	$\neg\underline{\boldsymbol{(xSy)}} \vee \{\neg\hat{S}, \hat{k}\}[x, y]$	:OR.2
S.**2**	$\neg\boldsymbol{(x\tilde{S}_1y)}^{\sharp} \vee\cdots\vee \neg\boldsymbol{(x\tilde{S}_ny)}^{\sharp} \vee \hat{\alpha}[x] \vee \hat{\alpha}[y]$	:MCC
S.**3**	$\neg\boldsymbol{(\underline{x}\tilde{S}_1y)}^{\sharp} \vee\cdots\vee \neg\boldsymbol{(\underline{x}\tilde{S}_ny)}^{\sharp} \vee \hat{\alpha}[x] \vee \hat{\alpha}[y]$	:⟦HC⟧: Lemma 5.2.11
S.4	$\neg\boldsymbol{(\underline{x}\tilde{S}_1y)}^{\sharp} \vee\cdots\vee \neg\boldsymbol{(\underline{y}\tilde{S}_nx)}^{\sharp} \vee \hat{\beta}[x] \vee \hat{\beta}[y]$	:HC

MCC[S.**2**]:	S.**2.1**	$\neg\boldsymbol{(x\tilde{S}_1y)}^{\sharp} \vee\cdots\vee \neg\boldsymbol{(x\tilde{S}_ny)}^{\sharp} \vee \hat{\alpha}[x] \vee u_{\hat{\alpha}}^{\tilde{S}_1..\tilde{S}_n}(y)$	
	S.**2.1.1**	$\neg\underline{\boldsymbol{(x\tilde{S}y)}} \vee \neg\hat{S}[x, y] \vee \hat{\alpha}[x] \vee u_{\hat{\alpha}}^{\tilde{S}_1..\tilde{S}_n}(y)$	:OR.2 ⇒ S.1
	S.**2.1.2**	$\neg\boldsymbol{(\underline{x}\tilde{S}_1y)}^{\sharp} \vee\cdots\vee \neg\boldsymbol{(\underline{x}\tilde{S}_ny)}^{\sharp} \vee \hat{\alpha}[x] \vee u_{\hat{\alpha}}^{\tilde{S}_1..\tilde{S}_n}(y)$	:⟦HC⟧: L. 5.2.11
	S.**2.1.3**	$\neg\boldsymbol{(\underline{x}\tilde{S}_1y)}^{\sharp} \vee\cdots\vee \neg\boldsymbol{(\underline{y}\tilde{S}_nx)}^{\sharp} \vee \hat{\alpha}[x] \vee u_{\hat{\alpha}}^{\tilde{S}_1..\tilde{S}_n}(y)$	:HC ⇒ S.4
	S.**2.2**	$\neg\boldsymbol{(x\tilde{S}_1y)}^{\sharp} \vee\cdots\vee \neg\boldsymbol{(x\tilde{S}_ny)}^{\sharp} \vee \neg u_{\hat{\alpha}}^{\tilde{S}_1..\tilde{S}_n}(x) \vee u_{\hat{\alpha}}^{\tilde{S}_1..\tilde{S}_n}(y)$	
	S.**2.2.1**	$\neg\underline{\boldsymbol{(xSy)}} \vee \neg\hat{S}[x, y] \vee \neg u_{\hat{\alpha}}^{\tilde{S}_1..\tilde{S}_n}(x) \vee u_{\hat{\alpha}}^{\tilde{S}_1..\tilde{S}_n}(y)$	:OR.2 ⇒ S.1
	S.**2.2.2**	$\neg\boldsymbol{(\underline{x}\tilde{S}_1y)}^{\sharp} \vee\cdots\vee \neg\boldsymbol{(\underline{x}\tilde{S}_ny)}^{\sharp} \vee \neg u_{\hat{\alpha}}^{\tilde{S}_1..\tilde{S}_n}(x) \vee u_{\hat{\alpha}}^{\tilde{S}_1..\tilde{S}_n}(y)$	:⟦HC⟧: L. 5.2.11
	S.**2.2.3**	$\neg\boldsymbol{(\underline{x}\tilde{S}_1y)}^{\sharp} \vee\cdots\vee \neg\boldsymbol{(\underline{y}\tilde{S}_nx)}^{\sharp} \vee \neg u_{\hat{\alpha}}^{\tilde{S}_1..\tilde{S}_n}(x) \vee u_{\hat{\alpha}}^{\tilde{S}_1..\tilde{S}_n}(y)$	:HC ⇒ S.4
	S.**2.3**	$\neg u_{\hat{\alpha}}^{\tilde{S}_1..\tilde{S}_n}(y) \vee \hat{\alpha}[y]$	:U

U	$\{\neg\hat{S}, \hat{S}, \hat{k}\}[h(x), x]$	
U.**1.1**	$\{\neg\hat{S}, \hat{S}, \hat{k}\}[h(x), x] \vee \underline{\boldsymbol{S[!h(x), x]}}^{\star}$	:OR.1
U.**1.2**	$\{\neg\hat{S}, \hat{S}, \hat{k}\}[h(x), x] \vee \underline{\boldsymbol{h(x)}}\boldsymbol{\tilde{S}(h(x)\mid x)}^{\star}$	:C
U.**1.3**	$[\![\, \{\neg\hat{S}, \hat{S}, \hat{k}\}[x] \vee \boldsymbol{S[x]}^{\sharp}]\!]$	:LP
U.**1.4**	$\hat{\beta}[x] \vee \underline{\boldsymbol{S[!x]}}^{\star}$	:OR.1
U.**1.5**	$\hat{\beta}[x] \vee \underline{\boldsymbol{x}}\boldsymbol{\tilde{S}x}^{\star}$	:C
OR[U.**1.1**; S.**1**]	$:\{\neg\hat{S}, \hat{S}, \hat{k}\}[h(x), x] \vee \{\neg\hat{S}, \hat{k}\}[!h(x), x]$	:U
OC[U.**1.2**; U.**1.2**]	$:\{\neg\hat{S}, \hat{S}, \hat{k}\}[h(x), x] \vee (h(x)\mid x)\tilde{S}(h(x)\mid x)$	:U
HC[{U.**1.2**}; S.**4**]	$:\{\neg\hat{S}, \hat{S}, \hat{k}\}[h(x), x]$	:U
LP[U.**1.3**]	$:\{\neg\hat{S}, \hat{S}, \hat{k}\}[x] \vee p_{S[\cdot]}(x)$	:U
	$:\neg p_{S[\cdot]}(x) \vee S[x]$	:U
OR[U.**1.4**; **1.1.3**]	$:\hat{\beta}[c] \vee \hat{\beta}[\hat{c}]$	:1
OR[U.**1.4**; **2.1.4**]	$:\hat{\beta}[\hat{f}(\overline{x}), \overline{x}] \vee \neg\hat{a}[!\overline{x}] \vee \hat{\alpha}[\overline{x}] \vee \hat{\beta}[!\hat{f}(\overline{x}), \overline{x}]$	:2
OC[U.**1.2**; U.**1.5**]	$:\{\neg\hat{S}, \hat{S}, \hat{k}\}[h(x), x] \vee x\tilde{S}(h(x)\mid x)$	:U
OC[U.**1.5**; U.**1.5**]	$:\hat{\beta}[x] \vee x\tilde{S}x$	:U

Continued on next page

U.**2**.**1** $\{\neg\hat{S},\hat{S},\hat{k}\}[h(x),x] \vee \boldsymbol{\neg}\underline{\boldsymbol{S[!h(x),x]}}$:OR.2
U.**2**.**2** $\{\neg\hat{S},\hat{S},\hat{k}\}[h(x),x] \vee \boldsymbol{\neg(}\underline{\boldsymbol{h(x)}}\boldsymbol{\tilde{S}(h(x)|x))}$:NC
U.**2**.**3** $\{\neg\hat{S},\hat{k}\}[x] \vee \boldsymbol{\neg}\underline{\boldsymbol{S[!x]}}$:OR.2
U.**2**.**4** $\{\neg\hat{S},\hat{k}\}[x] \vee \boldsymbol{\neg(xS}\underline{\boldsymbol{x}}\boldsymbol{)}$:NC.r
OR[U.**1**.**1**; U.**2**.**1**]: $\{\neg\hat{S},\hat{S},\hat{k}\}[h(x),x]$:U
OR[U.**1**.**4**; U.**2**.**1**]: $\hat{\beta}[x]$:U
NC[U.**1**.**2**; U.**2**.**2**]: $\{\neg\hat{S},\hat{S},\hat{k}\}[h(x),x] \vee \neg((h(x)|x)\tilde{S}(h(x)|x))$:U
OR[U.**1**.**1**; U.**2**.**3**]: $\{\neg\hat{S},\hat{k}\}[x]$:U
OR[U.**1**.**4**; U.**2**.**3**]: $\hat{\beta}[x]$:U
NC[U.**1**.**2**; U.**2**.**4**]: $\{\neg\hat{S},\hat{S},\hat{k}\}[h(x),x] \vee \neg(h(x)\tilde{S}(h(x)|x))$:U

U.**3**.**1** $\{\neg\hat{S},\hat{S},\hat{k}\}[h(x),x] \vee \underline{\boldsymbol{q[!h(x),x]}}^{\star}$:OR.1
U.**3**.**2** $\{\neg\hat{S},\hat{S},\hat{k}\}[h(x),x] \vee \boldsymbol{\neg}\underline{\boldsymbol{q[!h(x),x]}}$:OR.2
U.**3**.**3** $\{\neg\hat{S},\hat{k}\}[x] \vee \underline{\boldsymbol{q[!x]}}^{\star}$:OR.1
U.**3**.**4** $\{\neg\hat{S},\hat{k}\}[x] \vee \boldsymbol{\neg}\underline{\boldsymbol{q[!x]}}$:OR.2
OR[U.**3**.**1**; **2**.**2**.**1**]: $\{\neg\hat{S},\hat{S},\hat{k}\}[h(x),x] \vee \hat{\alpha}[h(x)]$:U
OR[**2**.**3**.**1**; U.**3**.**2**]: $\neg\hat{S}[!h(x)] \vee \hat{\alpha}[h(x)] \vee \{\neg\hat{S},\hat{S},\hat{k}\}[h(x),x]$:U
OR[U.**3**.**1**; U.**3**.**2**]: $\{\neg\hat{S},\hat{S},\hat{k}\}[h(x),x]$:U
OR[U.**3**.**3**; **1**.**1**.**2**]: $\{\neg\hat{S},\hat{k}\}[\hat{c}] \vee \hat{\beta}[\hat{c}]$:1
OR[**1**.**1**.**1**; U.**3**.**4**]: $\hat{\beta}[\hat{c}] \vee \{\neg\hat{S},\hat{k}\}[\hat{c}]$:1
OR[U.**3**.**3**; **2**.**1**.**2**]: $\{\neg\hat{S},\hat{k}\}[\hat{f}(\overline{x}),\overline{x}] \vee \neg\hat{a}[!\overline{x}] \vee \hat{\alpha}[\overline{x}] \vee \hat{\beta}[!\hat{f}(\overline{x}),\overline{x}]$:2
OR[**2**.**1**.**3**; U.**3**.**4**]: $\neg\hat{a}[!\overline{x}] \vee \hat{\alpha}[\overline{x}] \vee \hat{\beta}[!\hat{f}(\overline{x}),\overline{x}] \vee \{\neg\hat{S},\hat{k}\}[\hat{f}(\overline{x}),\overline{x}]$:2
OR[U.**3**.**3**; **2**.**2**.**1**]: $\{\neg\hat{S},\hat{k}\}[x] \vee \hat{\alpha}[x]$:U
OR[**2**.**3**.**1**; U.**3**.**4**]: $\neg\hat{S}[!x] \vee \hat{\alpha}[x] \vee \{\neg\hat{S},\hat{k}\}[x]$:U
OR[U.**3**.**3**; U.**3**.**4**]: $\{\neg\hat{S},\hat{k}\}[x]$:U

D.4 A Sketch of a Decision Procedure for the Guarded Fragment with Transitive Guards

In this section we sketch a procedure for the *guarded fragment with transitive guards and equality* $\mathcal{GF}_{\simeq}[\mathsf{TG}]$, based on the technique we have developed in this thesis. For deciding this fragment we will use the *subterm chaining calculus*, which is an extension of the *ordered chaining calculus* with special rules for equality (see System 7).

The decision procedure for $\mathcal{GF}_{\simeq}[\mathsf{TG}]$ that we sketch in this section is rather involved and we don't think it can be easily implemented. The main purpose of this section is to demonstrate the reasons behind decidability of $\mathcal{GF}_{\simeq}[\mathsf{TG}]$ and show how our techniques could be applied for designing a decision procedure for such a non-trivial fragment.

Difficulties with equality

If we go through the saturation strategy for $\mathcal{GF}[\mathsf{TG}]$ that we have proposed in subsection 5.2.1, we will notice that there is a case which does not directly extend to equality. A problem arises with clauses of form:

$$\mathsf{T}_{\simeq} \quad \neg\{!\hat{T}\}[!x, !y] \vee \hat{\alpha}[x] \vee \hat{\alpha}[y] \vee \boldsymbol{x \simeq y} \tag{D.1}$$

which are guarded by several transitive atoms and have no other atoms containing *both* x and y except for equality $x \simeq y$. Note that formulas of a similar form were essential in our undecidability proofs in the previous sections. For clauses of form (D.1) we can nigther apply the Transitive Closure rule, nor can we resolve on an atom containing both variables x and y, since the equality is the only such atom. In order to deal with clauses of this form, we introduce an extension of the Transitive Closure rule, but first we restrict such clauses using the following observation:

Proposition D.4.1. *T is a transitive relation iff T is a union of two transitive relations S and S such that (**i**) S is symmetric (that is a partial equivalence), (**ii**) S is an antisymmetric, and (**iii**) $S \circ S \subseteq S$, $S \circ S \subseteq S$.*

Proof. The "*if*" part follows from $(S{\cup}S){\circ}(S{\cup}S) \subseteq S \circ S{\cup}S{\circ}S{\cup}S{\circ}S{\cup}S{\circ}S \subseteq (S{\cup}S)$. To prove the "*only if*" part, let $S := T \cap T^{\smile}$ and $S := T \cap \neg(T^{\smile})$. Properties (i) – (iii) are easy to verify. ◫

This proposition allows us to consider only a restricted version of $\mathcal{GF}_{\simeq}[\mathsf{TG}]$, where each transivie atom is either symmetric or antysymmettic. Indeed, we can replace every subformula guarded with a transitive atom T in guards $\forall xy.(xTy \rightarrow F[x, y])$, $\exists x.(xTy \wedge F[x, y])$, etc. respectively with $\forall xy.(xSy \rightarrow F[x, y]) \wedge \forall x.(xSy \rightarrow$

$F[x,y])$, $\exists x.(xSy \wedge F[x,y]) \vee \exists x.(xSy \wedge F[x,y])$, etc.. This transformation preserves satisfiability of formulas by Proposition D.4.1 and produces clauses containing transitive guards that are either symmetric or antysymmetric. Now we can concentrate on clauses of form (D.1) where *all* guards are symmetric transitive relations, i.e., *partial equivalences*, since otherwise positive equation $x \simeq y$ can be removed, since $\neg xSx$ holds for every antysymmetric transitive relation S. The only price we need to pay for this, is to consider additional compositional axioms: $S \circ S \subseteq S$ and $S \circ S \subseteq S$, and clauses:

$$\neg xSy \vee ySx \textit{ (symmetry)} \qquad \textit{and} \qquad \neg xSy \vee \neg ySx \textit{ (antysymmetry)}$$

(In particular, the *symmetry* clause which does not fit our clause calss since it contains a transitive atom *positively*). It is easy to see that all compositional axioms remain associative, so we can apply the usual chaining rules. Note also that S can be obtained only from compositions with itself!

Symmetry can also be built into the chaining calculi by treating atoms xSy symmetrically. Alternatively, on can show that the Negative Chaining inferences with the *symmetry* clauses are redundant, which means that only Ordered Resolution can be applied to such clauses, which has the same effect as swapping the arguments in positive occurrences of symmetric atoms:

Lemma D.4.2. *Let N be a clause set containing the clauses:*

1. $C \vee \boldsymbol{sSt}$ $\qquad$ $\texttt{OR}[1;2]$: 3. $\quad C \vee tSs$
2. $\neg \boldsymbol{xSy}^{\sharp} \vee ySx$ $\quad$ *(Symmetry)*

where S is a transitive predicate symbol. Then the following Negative Chaining *inferences are redundant w.r.t. N:*

($\boldsymbol{a}$) $\texttt{NC}[1;2]$: 4_a. $\quad C \vee \neg(tSy) \vee ySs$;
($\boldsymbol{b}$) $\texttt{NC}[1;2]$: 4_b. $\quad C \vee \neg(xSs) \vee tSx$.

Proof. We prove the lemma for the case (a). The case (b) can be proved in the same way by swapping the arguments of S. W.l.o.g., we may assume that the clause 1 is ground.

The conclusion of every ground instance of inference ($\boldsymbol{a}$):

1. $C \vee \boldsymbol{sSt}$;
$2'$. $\neg \underline{\boldsymbol{s}}\boldsymbol{Sh}^{\sharp} \vee sSh$;
$\texttt{NC}[1;2']$: $4'_a$. $\quad C \vee \neg(tSh) \vee hSs$;

follows logically from the clause 3 and an instance of the *symmetry* clause:

$2''$. $\neg(tSh) \vee hS\underline{t}$; $\qquad$ 3. $C \vee \underline{t}Ss$;

since S is a transitive relation. Each of the clauses $2''$ and 3 are smaller than the clause $2'$ which can be shown using conditions of the Negative Chaining rule and conditions of admissible orderings. Therefore, the inference producing the clause 3_a is redundant. ◫

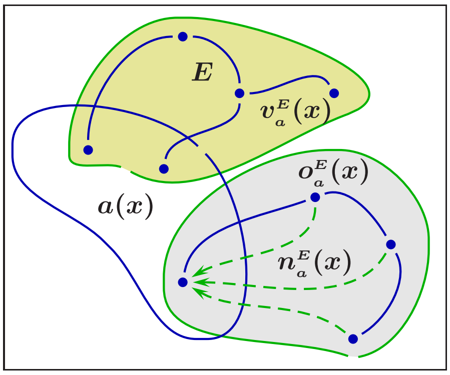
Since there is no difference in dealing with symmetry for transitive relations, we prefer to treat atoms xSy symmetrically in our clauses in order to avoid considering additional cases.

Auxiliary inference rules and redundancy

Let us try to understand the semantical meaning for clauses of the form:

$$\neg(xSy) \vee \neg a(x) \vee b(y) \vee x \simeq y \tag{D.2}$$

This clause says that for every element y that is S-reachable from some x with $a(x)$, we must have either $b(y)$ or $x \simeq y$. There are two situations possible for such y: (**1**) either x is *the only* element for y with these properties, or (**2**) there are at least two such elements $x_1 \not\simeq x_2$.


$\boldsymbol{E}$
$\boldsymbol{v_{a}^{E}(x)}$
$\boldsymbol{o_{a}^{E}(x)}$
$\boldsymbol{a(x)}$
$\boldsymbol{n_{a}^{E}(x)}$

In case (2), we must have $b(y)$ always, since otherwise we have $y \simeq x_1$ and $y \simeq x_2$ which is not possible, with $x_1 \not\simeq x_2$. In case (1) a S-equivalence class containing y must contain exactly one element x with $a(x)$.

Let us define a function $n_{a}^{S}(y)$ producing a S-*normal element w.r.t.* a such that (***i***) $n_{a}^{S}(y) \simeq x$ if S-equivalence class of y contains *exactly one* element x with $a(x)$ and (***ii***) $n_{a}^{S}(y) \simeq y$ otherwise. Furthermore, let $o_{a}^{S}(y)$ be an atom that holds on all elements y such that their equivalence class contains *exactly one* element x with $a(x)$, and, otherwise, $v_{a}^{S}(y)$ holds if there are *more than one* such elements. Then the following properties hold:

$\neg a(x) \vee o_{a}^{S}(x) \vee v_{a}^{S}(x)$ - *if $a(x)$ holds then either $v_{a}^{S}(x)$ or $o_{a}^{S}(x)$ holds*
$\neg(xSy) \vee \neg v_{a}^{S}(x) \vee v_{a}^{S}(y)$ - *the set of x where $v_{a}^{S}(x)$ holds and*
$\neg(xSy) \vee \neg o_{a}^{S}(x) \vee o_{a}^{S}(y)$ - *the set of x where $o_{a}^{S}(x)$ holds are colsed under S*
$\neg(xSy) \vee n_{a}^{S}(x) \simeq n_{a}^{S}(y)$ - *normal elements for equivalent elements are equal*
$\neg v_{a}^{S}(y) \vee b(y)$ - *if $v_{a}^{S}(y)$ holds then $b(y)$ holds*
$\neg o_{a}^{S}(x) \vee \neg a(x) \vee n_{a}^{S}(x) \simeq x$ - *if $a(x)$ and $o_{a}^{S}(x)$ hold then x is the normal of x*
$\neg o_{a}^{S}(y) \vee b(y) \vee n_{a}^{S}(y) \simeq y$ - *if $o_{a}^{S}(y)$ then either $b(y)$ or y is the normal of y*
$xS\, n_{a}^{S}(x) \vee n_{a}^{S}(x) \simeq x$ - *$n_{\alpha}^{S}(x)$ is either S-reachable from x or is equal to x*

This construction can be generalized to a simplification rule Equivalence Closure defined in Figure D.1. It is possible to show that one can always define atoms $v^S_\alpha(x)$, $o^S_\alpha(x)$ and function $n^S_\alpha(x)$ similarly as it is done above, such that all conclusions of this rules are `true` provided its premise is `true`. In other words:

Figure D.1 The Equivalence Closure rule

Equivalence Closure

$$\mathtt{EC}:\ \frac{\neg(\boldsymbol{xSy}) \vee \alpha[x] \vee \beta[y] \vee x \simeq y}{\begin{array}{ll} \alpha[x] \vee v^S_\alpha(x) \vee o^S_\alpha(x) & \neg v^S_\alpha(y) \vee \beta[y] \\ \neg(xSy) \vee \neg v^S_\alpha(x) \vee v^S_\alpha(y) & \neg o^S_\alpha(x) \vee \alpha[x] \vee n^S_\alpha(x) \simeq x \\ \neg(xSy) \vee \neg o^S_\alpha(x) \vee o^S_\alpha(y) & \neg o^S_\alpha(y) \vee \beta[y] \vee n^S_\alpha(y) \simeq y \\ \neg(xSy) \vee n^S_\alpha(x) \simeq n^S_\alpha(y) & xS\, n^S_\alpha(x) \vee n^S_\alpha(x) \simeq x \end{array}}$$

[*where (i) S is a symmetric transitive relation; (ii) v^S_α and o^S_α are extended unary predicate symbols and n^S_α is an extended unary functional symbol introduced for α and S .*]

Lemma D.4.3. *Equivalence Closure is a sound inference rule.*

We would not have introduced a new simplification rule (especially such complicated one) if this did not give us some advantage in saturation procedures. And indeed, Equivalence Closure helps avoiding Negative Chaining inferences in a similar way as the Transitive Closure rule does.

For proving redundancy of the Negative Chaining inferences with the premise of this rule (which will be essentially in the same way as for Transitive Closure), we need $n^S_\alpha(x)$ to behave more like an atom, rather than a function. In particular, we want that $(xSy) \succ n^S_\alpha(x) \simeq n^S_\alpha(y)$. So, we assume that the ordering on atoms and non-equational terms *respects the arity* of symbols. Unfortunately, such ordering is not admissible for the *subterm chaining calculus*, since it violates condition (C3) from Definition 3.5.4, when the first literal is equational (recall that equality $\simeq$ is a compositional relation): for example, we have $n^S_\alpha(x) \simeq n^S_\alpha(y) \not\succ \neg(xSn^S_\alpha(y))$. However, condition (C3) is only needed for the Compositional Resolution rule [see Kazakov, 2005]. Hence we can accept our assumption on $\succ$ provided that we do not use this rule in our procedure.

Lemma D.4.4. *Let N be a clause set containing the clauses:*

1. $C \vee \boldsymbol{sSt}$;

$\mathtt{R}_0$. $\neg(\boldsymbol{xSy}) \vee \alpha[x] \vee \beta[y] \vee x \simeq y$;

for some partial equivalence relation S, with the following conclusions of the Equivalence Closure inference:

$$\begin{array}{lll}
\mathtt{EC}[\mathtt{R}_0]: & \mathtt{R}_1.\ \alpha[x]\vee v^s_\alpha(x)\vee o^s_\alpha(x); & \mathtt{R}_5.\ \neg v^s_\alpha(y)\vee\beta[y];\\
 & \mathtt{R}_2.\ \neg(\boldsymbol{xSy})\vee\neg v^s_\alpha(x)\vee v^s_\alpha(y); & \mathtt{R}_6.\ \neg o^s_\alpha(x)\vee\alpha[x]\vee n^s_\alpha(x)\simeq x;\\
 & \mathtt{R}_3.\ \neg(\boldsymbol{xSy})\vee\neg o^s_\alpha(x)\vee o^s_\alpha(y); & \mathtt{R}_7.\ \neg o^s_\alpha(y)\vee\beta[y]\vee n^s_\alpha(y)\simeq y;\\
 & \mathtt{R}_4.\ \neg(\boldsymbol{xSy})\vee n^s_\alpha(x)\simeq n^s_\alpha(y); &
\end{array}$$

and the following ordered resolution inferences with them:

$$\begin{array}{ll}
\mathtt{OR}[1;\mathtt{R}_2]:\ 2.\quad C\vee\neg v^s_\alpha(s)\vee v^s_\alpha(t); & \mathtt{OR}[1;\mathtt{R}_4]:\ 4.\quad C\vee n^s_\alpha(s)\simeq n^s_\alpha(t);\\
\mathtt{OR}[1;\mathtt{R}_3]:\ 3.\quad C\vee\neg o^s_\alpha(s)\vee o^s_\alpha(t); &
\end{array}$$

Then the following Negative Chaining inferences are redundant w.r.t. N:

$$\begin{array}{llll}
(\boldsymbol{a}) & \mathtt{NC}[1;\mathtt{R}_0]:\ 5_a. & C\vee\neg(tSy)\vee\alpha[s]\vee\beta[y]\vee s\simeq y, & \textit{when } s\succ n^s_\alpha(t)\\
(\boldsymbol{b}) & \mathtt{NC}[1;\mathtt{R}_0]:\ 5_b. & C\vee\neg(xSs)\vee\alpha[x]\vee\beta[t]\vee x\simeq t, & \textit{when } t\succ n^s_\alpha(s)
\end{array}$$

Proof. Again, w.l.o.g. 1 is a ground clause.

($\boldsymbol{a}$) The conclusion of any ground instance of the first inference:

$$\begin{array}{l}
1.\ \ C\vee\underline{\boldsymbol{s}}\boldsymbol{St};\\
\mathtt{R}'_0.\ \neg(\underline{\boldsymbol{s}}\boldsymbol{Sh})\vee\alpha[s]\vee\beta[h]\vee s\simeq h;\\
\mathtt{NC}[1;\mathtt{R}'_0]:\ 5'_a.\quad C\vee\neg(tSh)\vee\alpha[s]\vee\beta[h]\vee s\simeq h
\end{array}$$

can be obtained from clauses 2 – 4:

$$\begin{array}{ll}
2.\ C\vee\neg\underline{v^s_\alpha(s)}\vee\dashuline{v^s_\alpha(t)}; & 4.\ C\vee\underline{\underline{n^s_\alpha(s)}}\simeq\underline{\underline{n^s_\alpha(t)}};\\
3.\ C\vee\neg\underline{o^s_\alpha(s)}\vee\dashuline{o^s_\alpha(t)}; &
\end{array}$$

and the following instances of clauses $\mathtt{R}_1$ – $\mathtt{R}_7$:

$$\begin{array}{ll}
\mathtt{R}'_1.\ \alpha[s]\vee\underline{v^s_\alpha(s)}\vee o^s_\alpha(s); & \mathtt{R}'_5.\ \neg\dotuline{v^s_\alpha(h)}\vee\beta[h];\\
\mathtt{R}'_2.\ \neg(tSh)\vee\neg\dashuline{v^s_\alpha(t)}\vee\dotuline{v^s_\alpha(h)}; & \mathtt{R}'_6.\ \neg\underline{o^s_\alpha(s)}\vee\alpha[s]\vee n^s_\alpha(s)\simeq s;\\
\mathtt{R}'_3.\ \neg(tSh)\vee\neg\dashuline{o^s_\alpha(t)}\vee\dotuline{o^s_\alpha(h)}; & \mathtt{R}'_7.\ \neg\dotuline{o^s_\alpha(h)}\vee\beta[h]\vee n^s_\alpha(h)\simeq h;\\
\mathtt{R}'_4.\ \neg(tSh)\vee\underline{\underline{n^s_\alpha(t)}}\simeq\underline{\underline{n^s_\alpha(h)}}; &
\end{array}$$

Indeed, by resolving on v^s_α clauses 2, $\mathtt{R}'_1$, $\mathtt{R}'_2$, $\mathtt{R}'_5$, we obtain the clause:

$$\mathtt{C}'_1.\ C\vee\neg(tSh)\vee\alpha[s]\vee\beta[h]\vee\underline{o^s_\alpha(s)};$$

Resolving this clause on o^s_α with clauses 3, $\mathtt{R}'_3$ and $\mathtt{R}'_7$, we obtain the clause:

$$\mathtt{C}'_2.\ C\vee\neg(tSh)\vee\alpha[s]\vee\beta[h]\vee\underline{\underline{n^s_\alpha(h)}}\simeq h;$$

Alternatively, the clause $\mathtt{C}'_1$ can be resolved with clause $\mathtt{R}'_6$ yielding:

$$\mathtt{C}'_3.\ C\vee\neg(tSh)\vee\alpha[s]\vee\beta[h]\vee\underline{\underline{n^s_\alpha(s)}}\simeq s;$$

Now using equality axioms and the clauses 4, $\mathtt{R}'_4$, $\mathtt{C}'_2$ and $\mathtt{C}'_3$ we obtain:

$$\mathtt{C}'_4.\ C\vee\neg(tSh)\vee\alpha[s]\vee\beta[h]\vee s\simeq h,$$

which is the same, as clause $5'_a$. It is easy to see that all clauses that have been used in the above inferences are smaller than clause $\mathtt{R}_0'$. This is essentially because all literals and terms resolved in the inferences are unary and contain no terms greater than s. So the inference that has produced clause $5_a'$ is redundant.

(***b***) The conclusion of any ground instance of the second inference:

$$\begin{array}{ll}
1. & C \vee \boldsymbol{sS\underline{t}}; \\
\mathtt{R}_0''. & \boldsymbol{\neg(hS\underline{t})} \vee \alpha[h] \vee \beta[t] \vee h \simeq t; \\
\mathtt{NC}[1;\mathtt{R}_0'']\text{: } 5_b'. & C \vee \neg(hSs) \vee \alpha[h] \vee \beta[t] \vee h \simeq t
\end{array}$$

can be similarly obtained from clauses 2 – 4 and the following instances of the clauses $\mathtt{R}_1$ – $\mathtt{R}_7$, all of them being smaller than clause $\mathtt{R}_0''$:

$$\begin{array}{ll}
\mathtt{R}_1''.\ \alpha[h] \vee \underline{v_\alpha^s(h)} \vee o_\alpha^s(h); & \mathtt{R}_5''.\ \neg\underline{v_\alpha^s(t)} \vee \beta[t]; \\
\mathtt{R}_2''.\ \neg(hSs) \vee \neg\underline{v_\alpha^s(h)} \vee \underline{v_\alpha^s(s)}; & \mathtt{R}_6''.\ \neg\underline{o_\alpha^s(h)} \vee \alpha[h] \vee n_\alpha^s(h) \simeq h; \\
\mathtt{R}_3''.\ \neg(hSs) \vee \neg\underline{o_\alpha^s(h)} \vee \underline{o_\alpha^s(s)}; & \mathtt{R}_7''.\ \neg\underline{o_\alpha^s(t)} \vee \beta[t] \vee n_\alpha^s(t) \simeq t; \\
\mathtt{R}_4''.\ \neg(hSs) \vee \underline{\underline{n_\alpha^s(h)}} \simeq \underline{\underline{n_\alpha^s(s)}}; &
\end{array}$$

$$\begin{array}{lll}
\mathtt{R}_1'', \mathtt{R}_2'', 2, \mathtt{R}_5'' & \vdash \mathtt{C}_1''. & C \vee \neg(hSs) \vee \alpha[h] \vee \beta[t] \vee \underline{o_\alpha^s(h)}; \\
\mathtt{C}_1'', \mathtt{R}_3'', 3, \mathtt{R}_7'' & \vdash \mathtt{C}_2''. & C \vee \neg(hSs) \vee \alpha[h] \vee \beta[t] \vee \underline{\underline{n_\alpha^s(t)}} \simeq t; \\
\mathtt{C}_1'', \mathtt{R}_6'' & \vdash \mathtt{C}_3''. & C \vee \neg(hSs) \vee \alpha[h] \vee \beta[t] \vee \underline{\underline{n_\alpha^s(h)}} \simeq h; \\
\mathtt{C}_3'', \mathtt{R}_4'', 4, \mathtt{C}_2'' & \vdash \mathtt{C}_4''. & C \vee \neg(hSs) \vee \alpha[h] \vee \beta[t] \vee h \simeq t,
\end{array}$$

Therefore, the inference producing clause $5_b'$ is redundant. ⧉

Redundancy of the Negative Chaining inferences with conclusions of the Equivalence Closure rule can be proven in a similar (but much simpler) way.

A saturation strategy for $\mathcal{GF}_{\simeq}[\mathsf{TG}]$

Note that the last conclusion of rule Equivalence Closure have not been used in Lemma D.4.4. This conclusion has a very special rôle that helps us to avoid growth of functional depth of clauses in inferences. The functional depth can grow, if we resolve, say, a clause of form $\neg p(x) \vee \underline{\boldsymbol{h(x)Sx}^\star}$, that originates from positive occurrences of S, with the 4-th conclusion of this rule:

$$1\ \neg p(x) \vee \underline{\boldsymbol{h(x)Sx}^\star} \qquad 2\ \underline{\boldsymbol{\neg xSy}^\sharp} \vee n_\alpha^s(x) \simeq n_\alpha^s(y)$$

$$\mathtt{OR}[1;2]\text{: } \neg p(x) \vee n_\alpha^s(h(x)) \simeq n_\alpha^s(x)$$

This gives us a clause with nested functional terms similar to what we have seen in subsection 4.3.3. However, this time, equational atoms with nested functional terms have a different form which might give problems (and actually does, if applied, say, to clauses for $\mathcal{GF}[\wedge\mathsf{TG}]$). This problem can be avoided using the last conclusion of the Equivalence Closure rule as follows. If we resolve this conclusion, with the 4-th conclusion of this rule but for a different premise we obtain:

$$1\ \boldsymbol{xSn^{s}_{\alpha_1}(x)}^{\star} \vee n^{S}_{\alpha_1}(x) \simeq x \qquad 2\ \neg\underline{\boldsymbol{xSy}}^{\sharp} \vee n^{S}_{\alpha}(x) \simeq n^{S}_{\alpha}(y)$$

$$\begin{array}{rll} \mathtt{OR}[1;2]\colon & n^{S}_{\alpha_1}(x) \simeq x \vee n^{S}_{\alpha}(n^{S}_{\alpha_1}(x)) \simeq n^{S}_{\alpha}(x)\colon & \text{which simplifies to:} \\ \Rightarrow\colon & n^{S}_{\alpha}(n^{S}_{\alpha_1}(x)) \simeq n^{S}_{\alpha}(x) \qquad\qquad : & \textit{(projection clause)} \end{array}$$

The clauses of the last form can be used in simplification to effectively reduce the depth of nested *normality functions* for the *same* relation E. Hence a special care needs to be taken in order to avoid appearance of nested functional terms for *different* E. To achieve this, we index every Skolem function $h(x)$ introduced for a symmetric transitive predicate symbol S with this symbol: $h^S(x)$. Skolem functions introduced for other transitive atoms are indexed with *any* symmetric transitive symbol S. It will be an invariant that whenever a partial equivalence S occurs *positively* in a clause, then all functional symbols in this clause must be indexed with S.

Taking this considerations into account, we define a clause class ($\mathbf{G}^{T}_{\simeq}$) for the guarded fragment with transitive guards and equality in Table D.3. Note that every clause of form `U` contains at most one symmetric transitive symbol S positively (possibly in several atoms) and all functional terms in such a clause must be indexed with S. Note that we may have three types of special predicate symbols S: symmetric transitive S, antisymmetric transitive S, equality $\simeq$ and antysymmetric P. The special symbols of the last type admit only compositional axioms

Table D.3 A clause class for the guarded fragment with transitive guards and equality

	Clause scheme	***Description***
($\mathbf{G}^{T}_{\simeq}$):	1 $\hat{\beta}[\hat{c}]$	*a ground clause containing compositional predicate symbols only negatively;*
	2 $\neg\hat{a}[!\overline{x}] \vee \alpha[\overline{x}] \vee \hat{\beta}[!\hat{f}(\overline{x}), \overline{x}]$	*a guarded clause whose extended atoms contain functional symbols;*
	`S` $\neg\{!\hat{S}\}[!x, !y] \vee \alpha[x] \vee \alpha[y]$	*an instance of the previous scheme where all literals containing different variables are compositional and occur negatively*
	`E` $\neg\{!\hat{S}\}[!x, !y] \vee \alpha[x] \vee \alpha[y] \vee x \simeq y$	*an instance of the scheme* 2 *where all literals containing different variables are either negative equivalence atoms or a positive equality*
	`U` $\{\neg\hat{S}, \hat{S}, \hat{P}, S, \simeq, \hat{k}^1\}[n^{S}_{\hat{\alpha}}(h^S(x)),$ $h^S(x), n^{S}_{\hat{\alpha}}(x), x]$	*clauses with one variable that may contain a Skolem function introduced for a compositional guard*
	`P` $n^{S}_{\hat{\alpha}}(n^{S}_{\hat{\alpha}_1}(x)) \simeq n^{S}_{\hat{\alpha}}(x)$	*projection clauses for extended functions*
	where $a := p \mid S \mid\ \simeq;\quad l := p \mid \neg p \mid \neg S \mid\ \simeq;\quad \alpha := \vee\{l\}$ $q := p \mid u^{S}_{\hat{\alpha}};\quad b := a \mid u^{S}_{\hat{\alpha}};\quad k := l \mid u^{S}_{\hat{\alpha}} \mid \neg u^{S}_{\hat{\alpha}};\quad \beta := \vee\{k\};\quad S := S \mid S \mid P \mid\ \simeq$	

with equality: $P \circ \simeq\ \subseteq\ \simeq$ and $\simeq \circ P\ \subseteq\ \simeq$. We introduce these symbols during translation of subformulas with positive occurrences of atoms S and S: formula $\forall x.[\mathtt{p}_F(x) \to \exists y.(xSy \wedge \mathtt{p}_{F_1}[x,y])]$ is first transformed to an equivalent formula $\forall x.[\mathtt{p}_F(x) \to \exists y.(xSy \wedge \mathtt{p}_{F_1}[x,y] \wedge x \not\simeq y) \vee (xEx \wedge \mathtt{p}_{F_1}[x,x])]$ and then is translated to clauses:

$$\begin{array}{lll} \neg\mathtt{p}_F(x) \vee xSh(x) \vee a(x) & \neg a(x) \vee xSx & \neg xPy \vee \mathtt{p}_{F_1}[x,y] \\ \neg\mathtt{p}_F(x) \vee xPh(x) \vee a(x) & \neg a(x) \vee \mathtt{p}_{F_1}[x,x] & \neg xPy \vee x \not\simeq y \end{array}$$

This trick is done to force superposition inferences into the *maximal argument* of atoms, in order to avoid growth of term depth (recall that superposition inferences into special atoms are done using the Ordered Subterm Chaining rule into their maximal argument). Note that non-special atoms in clauses of form U are *unary* which we indicated by writing k^1. The arguments of literals in a clause of form U could be only terms of forms $n^s_{\bar{\alpha}}(h^S(x))$, $h^S(x)$, $n^s_{\bar{\alpha}}(x)$ or x, where all functional symbol must be indexed with the same partial equivalence S, which is the only partial equivalence which may occur in this clause. In order to prevent from deep occurrences of functional symbols $h^S(x)$, we restrict an ordering $\succ$ further by requiring that $h^S(x) \succ n^s_{\bar{\alpha}}(x)$ for every projection function $n^s_{\bar{\alpha}}(x)$.

The saturation strategy for clause class $(\mathbf{G}^T_{\simeq})$ can be described as follows:

- *For clauses of forms* 1 *and* 2 *we apply a similar strategy as for the guarded fragment with equality: if a clause of form* 2 *contains a functional term, we produce inferences on its maximal literal, otherwise if this clause has a non-special guard, then we select one.*

- *If a clause of form* 2 *is non-functional and have only special guards, then we apply inferences on its maximal non-special literal, if it contains both variables* x *and* y*, or otherwise the maximal non-special literal can be simplified using the Literal Projection rule. In the remaining situation we must have a clause such that all of its non-special atoms are unary. If some of the guards in this clause is antysymmetric, then the equality* $x \simeq y$ *(if there is one) can be eliminated from such clause, and we obtain a clause of form* S*. Otherwise the clause must be of form* E*.*

- *For clauses of form* S *we apply a similar strategy as in the case with* $\mathcal{GF}[\mathsf{CG}]$*: we select all negative guards of this clause for the Negative Hyper-Chaining rule and make an application of such a rule redundant using Multi-Compositional Closure.*

- *For clauses of form* E*, we also select all negative guards. Now our strategy resembles the one for* $\mathcal{GF}[\mathsf{TG}]$ *because every function in* positive *occurrence of atom with* E *must be indexed with* E*: A Negative Hyper-Chaining inference between clauses of form* U *and a clause of form* E *can be applied either* (i) *on* *both variables x and y of this clauses in which case they are unified, and the inference produces a clause of form* U*, or (ii) on only one variable x or y, in which case the inference is possible if there is at most* one *guard, because it is not possible to apply inference on $xS_1\underline{h^{S_1}(x)}$ and $xS_2\underline{h^{S_2}(x)}$ with $S_1 \neq S_2$ since the functional terms must be unified. In case when there is only one guard in a clause of form* E*, the dangerous Negative Chaining inference can be avoided using the Equivalence Closure rule from Figure D.1.*

- *For clauses of form* U*, we apply the "divide and conquer" strategy similar to the one demonstrated for the guarded fragment with functional guards $\mathcal{GF}[\mathsf{FG}]$ in subsection 4.3.3: we separate all special atoms to different clauses using the Literal Projection rule. With this strategy we avoid Compositional Resolution inferences, which was required to make our ordering $\succ$ admissible.*
- *Finally we need to demonstrate that inferences between clauses of form* U *do not produce deeper clauses. Note that since all literals in such clauses are either special or unary, we apply the chaining and superposition inferences only on the maximal term of a clause. All such inferences will preserve the forms of arguments in such clauses, except when we superpose from a literal $\underline{h^S(x)} \simeq n^s_{\alpha_1}(x)$ to term $n^s_\alpha(\underline{h^S(x)})$ which produces term $n^s_\alpha(n^s_{\alpha_1}(x))$. Although, such terms are not allowed in clauses of form* U*, they can be immediately simplified to $n^s_\alpha(x)$ using a projection clause* P.

We see, that the resulted saturation decision procedure for $\mathcal{GF}_{\simeq}[\mathsf{TG}]$ is quite involved and uses almost all range our techniques: projection of literals, separation of binary literals, Multi-Compositional Closure and Equivalence Closure. The formal case analysis of possible inferences between clauses from $(\mathbf{G}^T_{\simeq})$ is too large and we do not provide it here. However we think that the above description of our procedure is convincing enouph to imply the following result:

Theorem D.4.5. *There is a saturation-based decision procedure for the guarded fragment with transitive guards and equality $\mathcal{GF}_{\simeq}[\mathsf{TG}]$, which can be implemented in 2EXPTIME.*

Bibliography

Allen, J. F. [1983], 'Maintaining knowledge about temporal intervals.', *Commun. ACM* **26**(11), 832–843. xxv, 158

Allen, J. F. [1991], 'Temporal reasoning and planning', pp. 1–67. 158

Andréka, H., van Benthem, J. & Németi, I. [1996], Modal languages and bounded fragments of predicate logic, Technical Report ML-1996-03, ILLC. vii, xi, 5, 75, 108

Armando, A., Ranise, S. & Rusinowitch, M. [2001], 'Uniform derivation of decision procedures by superposition', *Lecture Notes in Computer Science* **2142**, 513+. 97

Artale, A. & Franconi, E. [2000], 'A survey of temporal extensions of description logics.', *Ann. Math. Artif. Intell.* **30**(1-4), 171–210. 159

Baader, F. [1996], 'Using automata theory for characterizing the semantics of terminological cycles.', *Ann. Math. Artif. Intell.* **18**(2-4), 175–219. 10

Baader, F. [2002], Terminological cycles in a description logic with existential restrictions, LTCS-Report LTCS-02-02, Chair for Automata Theory, Institute for Theoretical Computer Science, Dresden University of Technology, Germany. See http://lat.inf.tu-dresden.de/research/reports.html. vi, x, 10, 11, 14, 28, 29

Baader, F. [2003], Restricted role-value-maps in a description logic with existential restrictions and terminological cycles, *in* 'Proceedings of the 2003 International Workshop on Description Logics (DL2003)', CEUR-WS. 6, 9, 11, 35, 164

Baader, F. & Nipkow, T. [1998], *Term Rewriting and All That*, Cambridge University Press, United Kingdom. 53, 57, 58, 59

Baader, F. & Nutt, W. [2003], Basic description logics., *in* Baader, Calvanese, McGuinness, Nardi & Patel-Schneider [2003], pp. 43–95. 14

Baader, F., Brandt, S. & Lutz, C. [2005], Pushing the $\mathcal{EL}$ envelope, *in* 'Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence IJCAI-05', Morgan-Kaufmann Publishers, Edinburgh, UK. vi, xi, 9, 11, 24, 30, 33, 34, 35, 43, 49, 192

Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D. & Patel-Schneider, P. F., eds [2003], *The Description Logic Handbook: Theory, Implementation, and Applications*, Cambridge University Press. 1, 64, 165, 265, 267, 269

Baader, F., Hladik, J., Lutz, C. & Wolter, F. [2003], From tableaux to automata for description logics., *in* M. Y. Vardi & A. Voronkov, eds, 'LPAR', Vol. 2850 of *Lecture Notes in Computer Science*, Springer, pp. 1–32. 3

Baaz, M., Egly, U. & Leitsch, A. [2001], Normal form transformations, *in* Robinson & Voronkov [2001], chapter 5, pp. 273–333. 92

Bachmair, L. & Ganzinger, H. [1990], On restrictions of ordered paramodulation with simplification, *in* 'Proceedings of the tenth international conference on Automated deduction', Springer-Verlag New York, Inc., pp. 427–441. v, vi, ix, x, 6, 10, 77, 80

Bachmair, L. & Ganzinger, H. [1994], 'Rewrite-based equational theorem proving with selection and simplification', *Journal of Logic and Computation* **4**(3), 217–247. v, vi, ix, x, 6, 10, 77

Bachmair, L. & Ganzinger, H. [1995], Ordered chaining calculi for first-order theories of binary relations, Technical Report MPI-I-95-2-009, Max-Planck-Institut für Informatik, Saarbrücken, Germany. Revised version to appear in JACM. vii, xii, 83, 155

Bachmair, L. & Ganzinger, H. [1998*a*], Equational reasoning in saturation-based theorem proving, *in* W. Bibel & P. Schmitt, eds, 'Automated Deduction — A Basis for Applications', Vol. I, Kluwer, chapter 11, pp. 353–397. 77

Bachmair, L. & Ganzinger, H. [1998*b*], 'Ordered chaining calculi for first-order theories of transitive relations', *Journal of the ACM*. Revised Version of MPI-I-95-2-009. vii, xii, 83, 84, 85, 155, 156, 169, 190

Bachmair, L. & Ganzinger, H. [2001], Resolution theorem proving, *in* Robinson & Voronkov [2001], chapter 2, pp. 19–99. 3, 40, 77, 90, 111

Bachmair, L., Ganzinger, H. & Waldmann, U. [1993*a*], Set constraints are the monadic class, *in* 'Eighth Annual IEEE Symposium on Logic in Computer Science', IEEE, Montreal, Canada, pp. 75–83. 72

Bachmair, L., Ganzinger, H. & Waldmann, U. [1993*b*], Superposition with simplification as a decision procedure for the monadic class with equality, *in* G. Gottlob, A. Leitsch & D. Mundici, eds, 'Computational Logic and Proof Theory, Third Kurt Gödel Colloquium, KGC'93', Vol. 713 of *Lecture Notes in Computer Science*, Springer, Brno, Czech Republic, pp. 83–96. 97, 114, 127, 226

Bachmair, L., Ganzinger, H., Lynch, C. & Snyder, W. [1995], 'Basic paramodulation', *Information and Computation* **121**(2), 172–192. Revised version of TR MPI-I-93-236, 1993. 142

Baldoni, M., Giordano, L. & Martelli, A. [1998], A tableau for multimodal logics and some (un)decidability results., *in* H. C. M. de Swart, ed., 'TABLEAUX', Vol. 1397 of *Lecture Notes in Computer Science*, Springer, pp. 44–59. 36, 64, 154

Baumgartner, P. & Tinelli, C. [2003], The model evolution calculus., *in* F. Baader, ed., 'CADE', Vol. 2741 of *Lecture Notes in Computer Science*, Springer, pp. 350–364. 195

Blackburn, P., de Rijke, M. & Venema, Y. [2001], *Modal logic*, Cambridge University Press, New York, NY, USA. 61

Börger, E., Grädel, E. & Gurevich, Y. [1997], *The Classical Decision Problem*, Perspectives of Mathematical Logic, Springer-Verlag. Second printing (Universitext) 2001. 72, 73, 74, 76, 77, 114, 119, 160

Borgida, A. [1996], 'On the relative expressiveness of description logics and predicate logics', *Artif. Intell.* **82**(1-2), 353–367. 32

Borgida, A., Lenzerini, M. & Rosati, R. [2003], Description logics for databases., *in* Baader, Calvanese, McGuinness, Nardi & Patel-Schneider [2003], pp. 462–484. 51

Boyer, R. S. [1971], Locking: A restriction of resolution, PhD thesis, University of Texas at Austin, Austin, TX. 111

Brachman, R. J. [1979], On the epistemological status of semantic networks, *in* N. V. Findler, ed., 'Associative Networks: Representation and Use of Knowledge by Computers', Academic Press, New York, pp. 3–50. Republished in Brachmann & Levesque [1985]. 64

Brachman, R. J. & Schmolze, J. G. [1985], 'An overview of the KL-ONE knowledge representation system.', *Cognitive Science* **9**(2), 171–216. 14, 35, 69, 70

Brachmann, R. J. & Levesque, H. J. [1985], *Readings in Knowledge Representation*, Morgan Kaufmann, Palo Alto.

Brandt, S. [2004*a*], Polynomial time reasoning in a description logic with existential restrictions, GCI axioms, and—what else?, *in* R. L. de Mantáras & L. Saitta, eds, 'Proceedings of the 16th European Conference on Artificial Intelligence (ECAI-2004)', IOS Press, pp. 298–302. 9, 11, 24, 29, 30, 35, 43, 49

Brandt, S. [2004*b*], Reasoning in $\mathcal{ELH}$ w.r.t. general concept inclusion axioms, LTCS-Report LTCS-04-03, Chair for Automata Theory, Institute for Theoretical Computer Science, Dresden University of Technology, Germany. See http://lat.inf.tu-dresden.de/research/reports.html. 41

Calvanese, D., Giacomo, G. D. & Franconi, E., eds [2003], *Proceedings of the 2003 International Workshop on Description Logics (DL2003), Rome, Italy September 5-7, 2003*, Vol. 81 of *CEUR Workshop Proceedings*. 273, 274

Chagrov, A. & Zakharyaschev, M. [1997], *Modal Logic*, Oxford University Press, Oxford. 61

Cohn, A. G., Bennett, B., Gooday, J. & Gotts, N. M. [1997], 'Qualitative spatial representation and reasoning with the region connection calculus.', *GeoInformatica* **1**(3), 275–316. 157

Dantsin, E., Eiter, T., Gottlob, G. & Voronkov, A. [2001], 'Complexity and expressive power of logic programming', *ACM Comput. Surv.* **33**(3), 374–425. 26, 32, 51

de Nivelle, H. [1995], Ordering Refinements of Resolution, PhD thesis, Technische Universiteit Delft. 40, 97, 99, 108, 111

de Nivelle, H. [1998], A resolution decision procedure for the guarded fragment, *in* C. Kirchner & H. Kirchner, eds, 'Proceedings of the 15th International Conference on Automated Deduction (CADE-14)', Vol. 1421 of *Lecture Notes in Artificial Intelligence*, Springer Verlag, Lindau, Germany, pp. 191–204. 60, 99, 100, 101, 108, 127

de Nivelle, H. [1999], Translation of S4 and K4 into GF and 2VAR, Unpublished: can be found on `http://www.mpi-sb.mpg.de/~nivelle`. 8, 153, 166, 167, 195

de Nivelle, H. [2000*a*], 'Deciding the E-plus class by an a posteriori, liftable order', *Annals of Pure and Applied Logic* **88**(1), 219–232. 97, 99, 101, 111

de Nivelle, H. [2000*b*], An overview of resolution decision procedures, *in* M. Faller, S. Kaufmann & M. Pauly, eds, 'Formalizing the Dynamics of Information', Vol. 91 of *CSLI Publications*, Center for the Study of Language and Information, Stanford University, Palo Alto, USA, pp. 115–130. 108, 111 de Nivelle, H. [2001], Splitting through new proposition symbols., *in* Nieuwenhuis & Voronkov [2001], pp. 172–185. 40

de Nivelle, H. & de Rijke, M. [2003], 'Deciding the guarded fragments by resolution', *Journal of Symbolic Computation* **35**, 21–58. 60, 97, 99, 100, 107, 108, 226, 236

de Nivelle, H. & Pratt-Hartmann, I. [2001], A resolution-based decision procedure for the two-variable fragment with equality., *in* T. N. R. Goré, A. Leitsch, ed., 'In: Proc. 1st Int. Joint Conf. on Automated Reasoning (IJCAR-2001)', Vol. 2083 of *Lect. Notes Artif. Intell.*, Springer, Berlin, pp. 211–225. 97, 111, 226

del Cerro, L. F. & Panttonen, M. [1988], 'Grammar logics', *Logique et Analyse* **121-122**, 123–134. 64, 154

Demri, S. [2001], 'The complexity of regularity in grammar logics and related modal logics.', *J. Log. Comput.* **11**(6), 933–960. vii, xii, 64, 154

Demri, S. & de Nivelle, H. [2005], 'Deciding regular grammar logics with converse through first-order logic', *Journal of Logic, Language and Information.* To appear in a special issue dedicated to guarded logics. 154, 155

Donini, F. M. [2003], Complexity of reasoning., *in* Baader, Calvanese, McGuinness, Nardi & Patel-Schneider [2003], pp. 96–136. v, ix, 3, 6, 35, 69, 164

Fermüller, C. G. & Salzer, G. [1993], Ordered paramodulation and resolution as decision procedure., *in* A. Voronkov, ed., 'LPAR', Vol. 698 of *Lecture Notes in Computer Science*, Springer, pp. 122–133. 97

Fermüller, C. G., Leitsch, A., Hustadt, U. & Tammet, T. [2001], Resolution decision procedures, *in* Robinson & Voronkov [2001], chapter 25, pp. 1791–1849. v, x

Fermüller, C., Leitsch, A., Tammet, T. & Zamov, N. [1993], *Resolution Methods for the Decision Problem*, Vol. 679 of *LNAI*, Springer, Berlin, Heidelberg. 11, 49, 59, 60, 97, 99, 108, 111, 226

Fitting, M. [1996], *First-Order Logic and Automated Theorem Proving. Second Edition*, Springer. 53

Gabbay, D. M. [1981], Expressive functional completeness in tense logic (preliminary report), *in* U. Mönnich, ed., 'Aspects of Philosophical Logic: Some Logical Forays into Central Notions of Linguistics and Philosophy', Reidel, Dordrecht, pp. 91–117. 74

Gabelaia, D., Kontchakov, R., Kurucz, A., Wolter, F. & Zakharyaschev, M. [2003], On the computational complexity of spatio-temporal logics., *in* I. Russell & S. M. Haller, eds, 'FLAIRS Conference', AAAI Press, pp. 460–464. 159

Ganzinger, H. & de Nivelle, H. [1999], A superposition decision procedure for the guarded fragment with equality, *in* 'Proc. 14th IEEE Symposium on Logic in Computer Science', IEEE Computer Society Press, pp. 295–305. vii, xi, 5, 7, 97, 99, 104, 108, 127, 128, 130, 150, 226, 236

Ganzinger, H. & Korovin, K. [2003], New directions in instantiation-based theorem proving, *in* 'Proc. 18th IEEE Symposium on Logic in Computer Science', IEEE Computer Society Press, pp. 55–64. 195

Ganzinger, H. & McAllester, D. A. [2001], A new meta-complexity theorem for bottom-up logic programs, *in* 'IJCAR '01: Proceedings of the First International Joint Conference on Automated Reasoning', Springer-Verlag, pp. 514–528. 52

Ganzinger, H. & McAllester, D. A. [2002], Logical algorithms., *in* P. J. Stuckey, ed., 'ICLP', Vol. 2401 of *Lecture Notes in Computer Science*, Springer, pp. 209–223. 52

Ganzinger, H., Hustadt, U., Meyer, C. & Schmidt, R. A. [2001], A resolution-based decision procedure for extensions of K4, *in* M. Zakharyaschev, K. Segerberg, M. de Rijke & H. Wansing, eds, 'Advances in Modal Logic, Volume 2', Vol. 119 of *CSLI Lecture Notes*, CSLI, Stanford, USA, chapter 9, pp. 225–246. 97, 169, 170, 171, 226

Ganzinger, H., Meyer, C. & Veanes, M. [1999], The two-variable guarded fragment with transitive relations, *in* 'Proc. 14th IEEE Symposium on Logic in Computer Science', IEEE Computer Society Press, pp. 24–34. 8, 36, 75, 152, 160, 161, 187, 189, 194

Gödel, K. [1933*a*], 'Eine interpretation des intuitionistischen aussagenkalküls', *Ergebnisse eines mathematisches Kolloquiums* **4**, 34–40. 61

Gödel, K. [1933*b*], 'Zum entscheidungsproblem des logischen funktionenkalküls', *Monatshefte Math. Phys.* **40**, 433–443. 73

Goldblatt, R. [1987], *Logics of time and computation*, Center for the Study of Language and Information. 61

Goldfarb, W. [1984], 'The unsolvability of the gödel class with identity', *Journal of Symbolic Logic* **49**, 1237–1252. 73

Goncalves, M.-E. & Grädel, E. [2000], Decidability issues for action guarded logics., *in* F. Baader & U. Sattler, eds, 'Description Logics', Vol. 33 of *CEUR Workshop Proceedings*, pp. 123–132. 144

Grädel, E. [1999], 'On the restraining power of guards', *Journal of Symbolic Logic* **64(4)**, 1719–1742. vii, xi, xii, 5, 6, 8, 75, 107, 108, 127, 130, 131, 132, 134, 137, 138, 160, 161, 162, 189, 193, 194, 236

Grädel, E. & Otto, M. [1999], 'On logics with two variables', *Theor. Comput. Sci.* **224**(1-2), 73–113. 74

Grädel, E. & Walukiewicz, I. [1999], Guarded fixed point logic, *in* 'Proceedings of 14th IEEE Symposium on Logic in Computer Science LICS '99, Trento', pp. 45–54. 75

Grädel, E., Kolaitis, P. & Vardi, M. [1997], 'On the Decision Problem for Two-Variable First-Order Logic', *Bulletin of Symbolic Logic* **3**, 53–69. 74, 114, 119, 238

Grädel, E., Otto, M. & Rosen, E. [1997], Two-Variable Logic with Counting is Decidable, *in* 'Proceedings of 12th IEEE Symposium on Logic in Computer Science LICS '97, Warschau'. 74, 138

Grosof, B. N., Horrocks, I., Volz, R. & Decker, S. [2003], Description logic programs: combining logic programs with description logic, *in* 'WWW '03: Proceedings of the twelfth international conference on World Wide Web', ACM Press, pp. 48–57. 51

Haarslev, V. & Möller, R. [2001], RACER system description, *in* 'IJCAR '01: Proceedings of the First International Joint Conference on Automated Reasoning', Springer-Verlag, pp. 701–706. 14, 44, 71

Haarslev, V., Lutz, C. & Möller, R. [1998], Foundations of spatioterminological reasoning with description logics., *in* 'KR', pp. 112–123. 159

Halpern, J. Y. & Shoham, Y. [1991], 'A propositional modal logic of time intervals.', *J. ACM* **38**(4), 935–962. 159

Hirsch, R. [1997], 'Expressive power and complexity in algebraic logic.', *J. Log. Comput.* **7**(3), 309–351. 159

Hladik, J. [2002], Implementation and optimisation of a tableau algorithm for the guarded fragment., *in* U. Egly & C. G. Fermüller, eds, 'TABLEAUX', Vol. 2381 of *Lecture Notes in Computer Science*, Springer, pp. 145–159. 148

Hopcroft, J. E. & Ullman, J. D. [1979], *Introduction to Automata Theory, Languages and Computation.*, Addison-Wesley. 36, 155

Horrocks, I. [1998], Using an expressive description logic: FaCT or fiction?, *in* 'Principles of Knowledge Representation and Reasoning:Proceedings 6th International Conference (KR'98)', Morgan Kauffman, pp. 636–647. ISBN 1558605541. 14, 71

Horrocks, I. & Patel-Schneider, P. F. [2004], 'Reducing OWL entailment to description logic satisfiability.', *J. Web Sem.* **1**(4), 345–357. 69, 71, 194

Horrocks, I. & Sattler, U. [2001], Ontology reasoning in the shoq(d) description logic., *in* B. Nebel, ed., 'IJCAI', Morgan Kaufmann, pp. 199–204. 131

Horrocks, I. & Sattler, U. [2004], 'Decidability of SHIQ with complex role inclusion axioms.', *Artif. Intell.* **160**(1-2), 79–104. 69, 154, 155

Horrocks, I. & Sattler, U. [2005], A tableaux decision procedure for SHOIQ, *in* 'Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)'. To appear. v, ix, 149

Horrocks, I., Sattler, U. & Tobies, S. [2000], 'Practical reasoning for very expressive description logics', *Logic Journal of the IGPL* **8**(3), 239–263. v, ix, 3, 14, 15, 41, 50, 69, 71, 151, 164

Hughes, G. & Cresswell, M. [1996], *A New Introduction to Modal Logic*, Routledge, London. 61

Hustadt, U. [1999], Resolution-Based Decision Procedures for Subclasses of First-Order Logic, PhD thesis, Universität des Saarlandes, Saarbrücken, Germany. 11, 97, 109, 111, 112

Hustadt, U. & Schmidt, R. A. [1999], Maslov's class K revisited, *in* H. Ganzinger, ed., 'Automated Deduction—CADE-16', Vol. 1632 of *Lecture Notes in Artificial Intelligence*, Springer, pp. 172–186. 97, 108, 111, 112, 226

Hustadt, U., Motik, B. & Sattler, U. [2004], A decomposition rule for decision procedures by resolution-based calculi., *in* F. Baader & A. Voronkov, eds, 'LPAR', Vol. 3452 of *Lecture Notes in Computer Science*, Springer, pp. 21–35. 50, 97, 142

Hustadt, U., Schmidt, R. A. & Weidenbach, C. [1999], MSPASS: Subsumption testing with SPASS, *in* P. Lambrix, A. Borgida, M. Lenzerini, R. Möller & P. Patel-Schneider, eds, 'Proc. of Intern. Workshop on Description Logics'99', Linköping University, pp. 136–137. 15, 50

Joyner Jr., W. H. [1976], 'Resolution strategies as decision procedures', *Journal of the ACM* **23**(3), 398–417. v, x, 11, 49, 97, 98, 114, 226

Kamin, S. & Lévy, J.-J. [1980], Two generalizations of the recursive path ordering, University of Illinois at Urbana-Chapaign. Unpublished manuscript. 57

Kazakov, Y. [2004], A polynomial translation from the two-variable guarded fragment with number restrictions to the guarded fragment., *in* J. J. Alferes & J. A. Leite, eds, 'JELIA', Vol. 3229 of *Lecture Notes in Computer Science*, Springer, pp. 372–384. 8, 9, 147, 148, 149, 193

Kazakov, Y. [2005], A framework of refutational theorem proving for saturation-based decision procedures, Research Report MPI-I-2005-2-004, Max-Planck-Institut für Informatik, Saarbrücken, Germany. 53, 58, 59, 60, 77, 78, 80, 83, 85, 86, 92, 171, 259

Kazakov, Y. & de Nivelle, H. [2003], Subsumption of concepts in $\mathcal{FL}_0$ for (cyclic) terminologies with respect to descriptive semantics is PSPACE-complete., *in* Calvanese, Giacomo & Franconi [2003]. 8, 10

Kazakov, Y. & de Nivelle, H. [2004], A resolution decision procedure for the guarded fragment with transitive guards., *in* D. A. Basin & M. Rusinowitch, eds, 'IJCAR', Vol. 3097 of *Lecture Notes in Computer Science*, Springer, pp. 122–136. 8, 9, 169

Kieronski, E. [2003], The two-variable guarded fragment with transitive guards is 2EXPTIME-hard, *in* A. D. Gordon, ed., 'FoSSaCS', Vol. 2620 of *Lecture Notes in Computer Science*, Springer, pp. 299–312. 75, 152, 161, 177

Kieronski, E. & Otto, M. [2005], Small substructures and decidability issues for two-variable first-order logic, *in* 'Proceedings of 20th IEEE Symposium on Logic in Computer Science LICS '05, Chicago, USA'. To appear. 163, 187, 195

Knuth, D. E. & Bendix, P. B. [1970], Simple word problems in universal algebras., *in* J. Leech, ed., 'Computational Problems in Abstract Algebra', Pergamon Press, Oxford, U. K., pp. 263–297. 57

Kripke, S. A. [1959], 'A completeness theorem in modal logic', *J. of Symbolic Logic* **24**, 1–14. 62

Kutz, O., Wolter, F., Sturm, H., Suzuki, N.-Y. & Zakharyaschev, M. [2003], 'Logics of metric spaces', *ACM Trans. Comput. Logic* **4**(2), 260–294. 156, 183

Letz, R. & Stenz, G. [2001], Automated theorem proving proof and model generation with disconnection tableaux., *in* Nieuwenhuis & Voronkov [2001], pp. 142–156. 195

Levesque, H. J. & Brachman, R. J. [1987], ‘Expressiveness and tractability in knowledge representation and reasoning.’, *Computational Intelligence* **3**, 78–93. 4, 65

Löwenheim, L. [1915], ‘Über möglichkeiten im relativkalkül’, *Math. Annalen* **76**, 447–470. 72

Lutz, C. [2004], ‘Combining interval-based temporal reasoning with general tboxes.’, *Artif. Intell.* **152**(2), 235–274. 159

Lutz, C. & Wolter, F. [2004], Modal logics of topological relations, LTCS-Report LTCS-04-05, Chair for Automata Theory, Institute for Theoretical Computer Science, Dresden University of Technology, Germany. See http://lat.inf.tu-dresden.de/research/reports.html. 159

Lutz, C., Wolter, F. & Zakharyaschev, M. [2003], Resasoning about concepts and similarity., *in* Calvanese et al. [2003]. 157

Massacci, F. [2000], ‘Single step tableaux for modal logics.’, *J. Autom. Reasoning* **24**(3), 319–364. 70

McAllester, D. [2002], ‘On the complexity analysis of static analyses’, *J. ACM* **49**(4), 512–537. 11, 26, 28, 52

Minksy, M. L. [1961], ‘Recursive unsolvability of post’s problem of "tag" and other topics in theory of turing machines’, *The Annals of Mathematics* **74**(3), 437–455. 76

Mortimer, M. [1975], ‘On language with two variables’, *Zeitschr. f. math. Logic u. Gundlagen d. Math.* **21**, 135–140. 74

Nieuwenhuis, R. & Rubio, A. [2001], Paramodulation-based theorem proving, *in* Robinson & Voronkov [2001], chapter 7, pp. 371–443. 77, 83

Nieuwenhuis, R. & Voronkov, A., eds [2001], *Logic for Programming, Artificial Intelligence, and Reasoning, 8th International Conference, LPAR 2001, Havana, Cuba, December 3-7, 2001, Proceedings*, Vol. 2250 of *Lecture Notes in Computer Science*, Springer. 269, 273

Nonnengart, A. & Weidenbach, C. [2001], Computing small clause normal forms, *in* Robinson & Voronkov [2001], chapter 6, pp. 335–367. 92

Ohlbach, H. J. [1991], ‘Semantics-based translation methods for modal logics.’, *J. Log. Comput.* **1**(5), 691–746. 49

Ohlbach, H. J. [1996], Scan - elimination of predicate quantifiers., *in* M. A. McRobbie & J. K. Slaney, eds, 'CADE', Vol. 1104 of *Lecture Notes in Computer Science*, Springer, pp. 161–165. 63

Ohlbach, H. J. & Schmidt, R. A. [1997], 'Functional translation and second-order frame properties of modal logics.', *J. Log. Comput.* **7**(5), 581–603. 63

Ohlbach, H. J., Nonnengart, A., de Rijke, M. & Gabbay, D. M. [2001], Encoding two-valued nonclassical logics in classical logic., *in* Robinson & Voronkov [2001], pp. 1403–1486. 63, 74

Pacholski, L., Szwast, W. & Tendera, L. [2000], 'Complexity results for first-order two-variable logic with counting', *SIAM J. Comput.* **29**(4), 1083–1117. 74, 138

Pan, G., Sattler, U. & Vardi, M. Y. [2002], BDD-based decision procedures for K, *in* A. Voronkov, ed., 'CADE', Vol. 2392 of *Lecture Notes in Computer Science*, Springer, pp. 16–30. 195

Paramasivam, M. & Plaisted, D. A. [1998], 'Automated deduction techniques for classification in description logic systems', *J. Autom. Reason.* **20**(3), 337–364. 50

Rector, A. L. [2002], Analysis of propagation along transitive roles: Formalisation of the GALEN experience with medical ontologies, *in* I. Horrocks & S. Tessaris, eds, 'Description Logics', Vol. 53 of *CEUR Workshop Proceedings*. 6, 42, 69, 153

Riazanov, A. & Voronkov, A. [2001], Splitting without backtracking., *in* B. Nebel, ed., 'IJCAI', Morgan Kaufmann, pp. 611–617. 40

Riazanov, A. & Voronkov, A. [2002], 'The design and implementation of vampire', *AI Commun.* **15**(2), 91–110. 3, 50

Robinson, G. A. & Wos, L. [1969], Paramodulation and theorem proving in first order theories with equality, *in* Meltzer & Mitchie, eds, 'Machine Intelligence 4', Edinburg University Press. 80

Robinson, J. A. [1965], 'A machine-oriented logic based on the resolution principle', *Journal of the ACM* **12**(1), 23–41. 10, 78

Robinson, J. A. & Voronkov, A., eds [2001], *Handbook of Automated Reasoning (in 2 volumes)*, Elsevier and MIT Press. 266, 269, 274, 277

Sagonas, K. F., Swift, T. & Warren, D. S. [1994], XSB as an efficient deductive database engine, *in* R. T. Snodgrass & M. Winslett, eds, 'Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data, Minneapolis, Minnesota, May 24-27, 1994', ACM Press, pp. 442–453. 11, 43

Sattler, U. & Vardi, M. Y. [2001], The hybrid mu-calculus, *in* R. Goré, A. Leitsch & T. Nipkow, eds, ‘Proceedings of the International Joint Conference on Automated Reasoning’, Vol. 2083 of *LNAI*, Springer Verlag, pp. 76–91. 131

Schild, K. [1991], A correspondence theory for terminological logics: Preliminary report., *in* ‘IJCAI’, pp. 466–471. 49, 66, 70

Schmidt, R. A. [1997], Optimised Modal Translation and Resolution, PhD thesis, Universität des Saarlandes, Saarbrücken, Germany. 63, 97

Schmidt, R. A. & Hustadt, U. [2003], Mechanised reasoning and model generation for extended modal logics, *in* H. C. M. de Swart, E. Orlowska, G. Schmidt & M. Roubens, eds, ‘Theory and Applications of Relational Structures as Knowledge Instruments’, Vol. 2929 of *Lecture Notes in Computer Science*, Springer, pp. 38–67. Survey paper commissioned for the Kickoff Volume of COST Action 274. 11, 15, 109

Schmidt-Schauß, M. [1989], Subsumption in KL-ONE is undecidable., *in* ‘KR’, pp. 421–431. vii, xii, 6, 35, 69, 70, 164

Schmidt-Schauß, M. & Smolka, G. [1991], ‘Attributive concept descriptions with complements.’, *Artif. Intell.* **48**(1), 1–26. 14, 65, 66, 70

Schulz, S. & Hahn, U. [2001], Parts, locations, and holes - formal reasoning about anatomical structures., *in* S. Quaglini, P. Barahona & S. Andreassen, eds, ‘AIME’, Vol. 2101 of *Lecture Notes in Computer Science*, Springer, pp. 293–303. 6, 154

Scott, D. [1962], ‘A decision method for validity of sentences in two variables’, *Journal of Symbolic Logic* **27**, 377. 74

Spackman, K. A. [2000], ‘Managing clinical terminology hierarchies using algorithmic calculation of subsumption: Experience with SNOMED-RT’, *J. of the American Medical Informatics Association.* Fall Symposium Special Issue. 69

Spackman, K. A. [2001], ‘Normal forms for description logic expressions of clinical concepts in SNOMED RT’, *J. of the American Medical Informatics Association* pp. 627–631. Symposium Supplement. 29

Spackman, K. A., Campbell, K. E. & Cote, R. A. [1997], ‘SNOMED RT: A reference terminology for health care’, *J. of the American Medical Informatics Association* pp. 640–644. Fall Symposium Supplement. 10, 42

Szwast, W. & Tendera, L. [2001], On the decision problem for the guarded fragment with transitivity., *in* ‘LICS’, pp. 147–156. 8, 75, 152, 153, 161, 163, 185, 188, 194

Tammet, T. [1990], The resolution program, able to decide some solvable classes, *in* P. Martin-Löf & G. Mints, eds, 'Proceedings of the International Conference on Computer Logic (COLOG-88)', Vol. 417 of *LNCS*, Springer, pp. 300–312. 97, 99, 108, 111

Tobies, S. [2000], 'The complexity of reasoning with cardinality restrictions and nominals in expressive description logics.', *J. Artif. Intell. Res. (JAIR)* **12**, 199–217. 149

Tobies, S. [2001], Complexity Results and Practical Algorithms for Logics in Knowledge Representation, PhD thesis, RWTH Aachen, Germany. v, ix, 3, 15, 147, 148

Tsarkov, D., Riazanov, A., Bechhofer, S. & Horrocks, I. [2004], Using vampire to reason with OWL., *in* S. A. McIlraith, D. Plexousakis & F. van Harmelen, eds, 'International Semantic Web Conference', Vol. 3298 of *Lecture Notes in Computer Science*, Springer, pp. 471–485. 50, 52

Vardi, M. [1996], Why is modal logic so robustly decidable?, *in* N. Immerman & P. G. Kolaitis, eds, 'Descriptive Complexity and Finite Models', Vol. 31 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, American Mathematical Society, Princeton University, pp. 149–184. 75

Waldmann, U. [1997], Cancellative Abelian Monoids in Refutational Theorem Proving, PhD thesis, Universität des Saarlandes. 97

Wang, H. [1961], 'Proving theorems by pattern recognition II', *Bell System Technical Journal* **40**, 1–41. 76

Weidenbach, C. [2001], Combining superposition, sorts and splitting, *in* Robinson & Voronkov [2001], chapter 27, pp. 1965–2013. 90

Weidenbach, C., Brahm, U., Hillenbrand, T., Keen, E., Theobalt, C. & Topić, D. [2002], SPASS version 2.0, *in* A. Voronkov, ed., 'Automated deduction, CADE-18 : 18th International Conference on Automated Deduction', Vol. 2392 of *Lecture Notes in Artificial Intelligence*, Springer, Kopenhagen, Denmark, pp. 275–279. 3

Wessel, M. [2001], Obstacles on the way to qualitative spatial reasoning with description logics: Some undecidability results., *in* C. A. Goble, D. L. McGuinness, R. Möller & P. F. Patel-Schneider, eds, 'Description Logics', Vol. 49 of *CEUR Workshop Proceedings*. 159

Wolter, F. & Zakharyaschev, M. [2003], Reasoning about distances, *in* G. Gottlob & T. Walsh, eds, 'Proceedings of the Eighteenth International Joint Conference on

Artificial Intelligence (IJCAI-03)’, Morgan Kaufmann, pp. 1275–1282. 156, 157, 195

Index