Efficient Reasoning Procedures for Complex First-Order Theories

Patrick Wischnewski

Dissertation zur Erlangung des Grades
des Doktors der Ingenieurwissenschaften
der Naturwissenschaftlich–Technischen Fakultäten
der Universität des Saarlandes

Saarbrücken
2012

Tag des Kolloquiums	6. November 2012
Dekan	Prof. Dr. Mark Groves
Vorsitzender des Prüfungsausschusses	Prof. Bernd Finkbeiner, Ph.D.
Berichterstatter	Prof. Dr. Torsten Schaub
	Prof. Dr. Christoph Weidenbach
	Prof. Dr. Gerhard Weikum
Akademischer Mitarbeiter	Dr. Mark Kaminski

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbstständig 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 20.03.2012

Abstract

The complexity of a set of first-order formulas results from the size of the set and the complexity of the problem described by its formulas.

Decision Procedures for Ontologies

This thesis presents new superposition based decision procedures for large sets of formulas. The sets of formulas may contain expressive constructs like transitivity and equality. The procedures decide the consistency of knowledge bases, called ontologies, that consist of several million formulas and answer complex queries with respect to these ontologies. They are the first superposition based reasoning procedures for ontologies that are at the same time efficient, sound, and complete.

The procedures are evaluated using the well-known ontologies YAGO, SUMO, and CYC. The results of the experiments, which are presented in this thesis, show that these procedures decide the consistency of all three above-mentioned ontologies and usually answer queries within a few seconds.

Reductions for General Automated Theorem Proving

Sophisticated reductions are important in order to obtain efficient reasoning procedures for complex, particularly undecidable problems because they restrict the search space of theorem proving procedures. In this thesis, I have developed a new powerful reduction rule. This rule enables superposition based reasoning procedures to find proofs in sets of complex formulas. In addition, it increases the number of problems for which superposition is a decision procedure.

Zusammenfassung

Die Komplexität einer Formelmenge für einen automatischen Theorembeweiser in Prädikatenlogik 1. Stufe ergibt sich aus der Anzahl der zu betrachtenden Formeln und aus der Komplexität des durch die Formeln beschriebenen Problems.

Entscheidungsprozeduren für Ontologien

Diese Arbeit entwickelt effiziente auf Superposition basierende Beweisprozeduren für sehr große entscheidbare Formelmengen, die ausdrucksstarke Konstrukte, wie Transitivität und Gleichheit, enthalten. Die Prozeduren ermöglichen es Wissenssammlungen, sogenannte Ontologien, die aus mehreren Millionen Formeln bestehen, auf Konsistenz hin zu überprüfen und Antworten auf komplizierte Anfragen zu berechnen. Diese Prozeduren sind die ersten auf Superposition basierten Beweisprozeduren für große, ausdrucksstarke Ontologien, die sowohl korrekt und vollständig, als auch effizient sind.

Die entwickelten Prozeduren werden anhand der weit bekannten Ontologien YAGO, SUMO und CYC evaluiert. Die Experimente zeigen, dass diese Prozeduren die Konsistenz aller untersuchten Ontologien entscheiden und Anfragen in wenigen Sekunden beantworten.

Reduktionen für allgemeines Theorembeweisen

Um effiziente Prozeduren für das Beweisen in sehr schwierigen und insbesondere in unentscheidbaren Formelmengen zu erhalten, sind starke Reduktionsregeln, die den Beweisraum einschränken, von essentieller Bedeutung. Diese Arbeit entwickelt eine neue mächtige Reduktionsregel, die es Superposition ermöglicht Beweise in sehr schwierigen Formelmengen zu finden und erweitert die Menge von Problemen, für die Superposition eine Entscheidungsprozedur ist.

Acknowledgments

First and foremost, I would like to thank my doctoral advisor, Prof. Dr. Christoph Weidenbach, for his persistent support, his guidance, and his always inspiring input. I appreciate being his doctoral student and am grateful for everything I have learned from him.

I thank my colleagues in the Automation of Logic group at the Max Planck Institute for Informatics for the inspiring working environment. Special thanks to Arnaud Fietzke for proofreading this thesis.

Also, I thank Prof. Dr. Gerhard Weikum for his expertise concerning YAGO and I thank him and Prof. Dr. Torsten Schaub for joining my examination board. I thank the Max Planck Research School for Computer Science for supporting my research.

Finally, I thank my parents, Claudia and Wolf-Dieter, my parents-in-law, Antonietta and Matteo, and my wife, Fabiana, for supporting and motivating me while working on this thesis.

Contents

Part I.

Decision Procedures for Ontologies

1. Introduction

Today, we can already find answers to many questions in the Internet with the assistance of search engines like Google, Bing and Yahoo. For a given query, they return webpages containing the keywords of the query. Searching for a precise answer to a query rather than webpages often reaches the limit of keyword based search engines. The problem is that the search engines do not understand the meaning of the words contained in both the webpages and the query. Therefore, in this thesis I present a query answering engine that computes precise answers to complex queries.

Consider the following query: "Which German physicist is also a politician?" On 17 January 2012 Google found 169,000,000 websites containing the keywords "German", "physicist" or "politician". The first website found is about climate politics, the second is about German physics and the third about Max Planck. The fourth hit is an article that appeared in The New York Times with the title "Merkel's Path: Brinkmanship for Debt Crisis", which contains the following sentence: "..., Mrs. Merkel, an East German physicist turned politician, ...". From this sentence we, as humans, can conclude that Angela Merkel is an answer to the above query.

Consequently, a search based on keywords is only successful if a document is found that contains the right answer. For example, reformulating the query to ask "Which German politician is a scientist?" did not find a webpage among the first ten results containing the name Angela Merkel.

The reason for this behavior is that the search engines neither understand the content of the webpages nor the meaning of the query. They rather search the webpages for the character strings composing the respective keywords. A website containing the keywords of the query does not necessarily contain the answer. If a computer understands the query and the knowledge contained in the website, it would return "Angela Merkel" as an answer in both of the above cases. Figure 1.1 depicts examples of queries that cannot be directly answered by keyword based search engines.

In order to accomplish that a computer understands the provided knowledge, the knowledge has to be represented in a way that can be interpreted by a computer. Such a representation is called an *ontology*. The representation of the knowledge in an ontology ranges from a rather informal representation to a representation in logic. Ontologies are briefly presented in the next section.

Answering complex queries in large ontologies precisely and completely requires a precise formal representation of the ontology and reasoning procedures that fulfill the following requirements: The reasoning procedures support a language expressive enough

Which politicians are also physicists?

Which predecessor of G.W. Bush has graduated from the same high school as his wife?

In which country was Angela Merkel born?

Which physicists were born in the same locations as all their children?

Which politician is also physicist and born in Europe?

Which successor of Helmut Schmidt is politician and physicist?

Does every German politician have a predecessor who is born in Germany?

Which politicians were born in Germany?

Who was the first German chancellor?

Figure 1.1.: Example queries

to represent the ontology as well as the query. Furthermore, the reasoning procedure is at the same time sound, complete, terminating, and practically feasible for ontologies consisting of several million formulas.

In this thesis, I present reasoning procedures that accomplish all these requirements.

1.1. Representing Knowledge in an Ontology

An ontology [Gru95, VHLPs08, Sta09] represents the knowledge of a particular domain. It consists of a collection of *facts* about *entities*. For example, people, cities, movies and machines are entities. Examples for facts are the relation between entities, "Albert Einstein was born in Ulm", the classification of entities, "Albert Einstein was a physicist", and subclass relations, "physicists are scientists".

Ontologies have traditionally been built manually by domain experts and ontology engineers. They differ in quality and in the amount of contained knowledge. The representation of the knowledge in an ontology varies from a rather informal representation to a mathematically precise formal representation using logics like first-order logic, higher-order logic or modal logic. The handbook on ontologies [Sta09] discusses the general design and maintenance of ontologies.

The following are examples for existing ontologies: YAGO [SKW07, SKW08], CYC [Len95], WordNet [Fel98]. DBPedia [ABK+07], SUMO [NP01a], KnowItAll [ECD+04], WikiTaxonomy [PS08], Omega [PHP08].

The main motivation for the work presented in this thesis was the development of first-order reasoning procedures for YAGO in order to verify its consistency and answer

arbitrary first-order queries. The YAGO ontology is automatically generated out of Wikipedia and WordNet [Fel98]. YAGO contains information about more than two million entities and has more than 15 million entries. A manual evaluation of YAGO by randomly choosing facts and comparing them with the respective Wikipedia page showed an accuracy of approximately 95% [SKW07]. The YAGO ontology is exceptional when compared to other ontologies because it is fully automatically generated, has a high coverage, and at the same time, a high accuracy rate.

The knowledge contained in YAGO is represented in a format that is a slight extension of the *Resource Description Format* (RDF) [Bec04, Sta09]. YAGO's knowledge is represented in triples of the following form:

$$\text{arg1} \quad \text{rel} \quad \text{arg2}\,.$$

This kind of triple is called a *fact*. For example, expressing "Albert Einstein was born in Ulm" is encoded as the fact

$$\text{AlbertEinstein} \quad \text{bornIn} \quad \text{Ulm}, \tag{1.1}$$

and the fact "Albert Einstein was of type physicist" is denoted as

$$\text{AlbertEinstein} \quad \text{type} \quad \text{physicist}\,. \tag{1.2}$$

In addition, YAGO formulates constraints like the functionality constraint for the relation bornIn

$$\text{bornIn} \quad \text{type} \quad \text{yagoFunction}, \tag{1.3}$$

and it also defines transitive relations, for example,

$$\text{locatedIn} \quad \text{type} \quad \text{yagoTransitiveRelation}\,. \tag{1.4}$$

This thesis defines a mathematically precise semantics for the YAGO ontology by translating YAGO into a representation in first-order logic. Verifying the consistency of YAGO and answering queries with the knowledge of YAGO both correspond to first-order reasoning tasks performed on the first-order representation of YAGO. A consistency check of YAGO verifies, for example, that all functionality constraints are fulfilled. Answering a query corresponds to checking whether the query logically follows from the knowledge of YAGO. The translation of YAGO and the respective reasoning tasks are depicted in the next section.

1.2. Reasoning in Ontologies

In order to define a mathematically precise semantics for YAGO, I present in Chapter 4 that YAGO can be automatically translated into a fragment of first-order logic, which I call BSH-Y2. The BSH-Y2 fragment is a subset of the Bernays–Schönfinkel Horn fragment with equality, which is a decidable fragment of first-order logic. The following

shows a brief overview of the translation. Additionally, it depicts the first-order reasoning problems which correspond to the operations for ontologies, namely consistency checking and query answering.

For example, fact 1.1 and fact 1.2 can, respectively, be represented by the following first-order formulas:

$$\text{bornIn}(\text{AlbertEinstein}, \text{Ulm})$$
$$\text{physicist}(\text{AlbertEinstein})$$

The constraint that bornIn is a functional relation is expressed by the following first-order formula:

$$\forall x, y, z(\text{bornIn}(x, y) \wedge \text{bornIn}(x, z) \rightarrow y \approx z),$$

where $\approx$ denotes equality. The fact that the relation locatedIn is transitive is stated by the formula

$$\forall x, y, z(\text{locatedIn}(x, y) \wedge \text{locatedIn}(y, z) \rightarrow \text{locatedIn}(x, z)).$$

The queries shown in Figure 1.1 can also be represented as first-order formulas. For example, the following query asks for "Physicists who were born in the same location as all their children":

$$\exists x, y(\text{physicist}(x) \wedge \text{bornIn}(x, y) \wedge \forall z.\, \text{hasChild}(x, z) \rightarrow \text{bornIn}(z, y)).$$

The next query asks "Does every German politician have a predecessor born in Germany?":

$$\forall x(\text{politicianOf}(x, \text{Germany}) \rightarrow \exists y, z(\text{hasSuccessor}(y, x) \wedge \text{bornIn}(y, z) \wedge \text{locatedIn}(z, \text{Germany}))).$$

The operations for ontologies, namely checking consistency and answering queries, correspond to first-order reasoning problems, which are depicted in the remainder of this section. A set of first-order formulas is called a theory. Assume, N is the theory from BSH-Y2 representing an ontology and Φ is a first-order formula called the query. Answering Φ in terms of the theory N corresponds to the reasoning task that verifies whether N implies Φ; written $N \models \Phi$. If N is inconsistent, denoted as $N \models \bot$, then Φ is trivially entailed by N because from a inconsistency everything is logically implied. In order to prevent the query answering procedure from returning trivial answers, one has to make sure that N is consistent, i.e., $N \not\models \bot$. In summary, before answering queries in N, one has to prove its satisfiability.

The standard first-order semantics for answering the query Φ with respect to N corresponds to an *open world assumption*. This is because $N \models \Phi$ means that Φ holds in all models of N; formally, for all models I with $I \models N$, it holds that $I \models \Phi$.

For example, consider the theory $N' = \{P(a), P(a) \rightarrow Q(a)\}$, the query $\Phi' = \forall x(P(x) \rightarrow Q(x))$, and the model $I' = \{P(a), Q(a), P(b)\}$. In this case, I' is a model of N', i.e., $I' \models N'$, but it is not a model of Φ'. So, $N' \not\models \Phi'$.

$\exists x(\text{politician}(x) \wedge \text{physicist}(x))$

$\exists x, y, z(\text{hasSuccessor}(x, \text{GeorgeWBush}) \wedge \text{graduatedFrom}(x, z) \wedge$
$\quad \text{graduatedFrom}(y, z) \wedge \text{isMarriedTo}(x, y))$

$\exists x, y(\text{bornIn}(\text{Angela\_Merkel}, y) \wedge \text{locatedIn}(x, y) \wedge \text{country}(y))$

$\exists x, y(\text{physicist}(x) \wedge \text{bornIn}(x, y) \wedge \forall z.\, \text{hasChild}(x, z) \rightarrow \text{bornIn}(z, y))$

$\exists x(\text{bornIn}(x, y) \wedge \text{politician}(x) \wedge \text{locatedIn}(x, \text{Europe}) \wedge \text{physicist}(x))$

$\exists x(\text{politician}(x) \wedge \text{physicist}(x) \wedge \text{hasSuccessor}(\text{Helmut\_Schmidt}, x))$

$\forall x(\text{politicianOf}(x, \text{Germany}) \rightarrow \exists y, z(\text{hasSuccessor}(y, x) \wedge \text{bornIn}(y, z) \wedge$
$\quad \text{locatedIn}(z, \text{Germany})))$

$\exists x(\text{politician}(x) \wedge \text{bornInCountry}(x, \text{Germany}))$

$\exists x(\text{GermanChancellor}(x) \wedge \forall y(\neg\, \text{hasPredecessor}(x, y) \vee \neg\, \text{GermanChancellor}(y)))$

Figure 1.2.: Queries in first-order logic

In contrast to the standard first-order semantics, the minimal model semantics corresponds to a *closed world assumption*. This means that the query Φ holds if it is implied only by the minimal model N_I of N, i.e., $N_I \models \Phi$. The model N_I is minimal in terms of set inclusion. Note, for each set of formulas from BSH-Y2, there is a unique minimal model. In the above example, the minimal model of N' is $N'_I = \{P(a), Q(a)\}$. This is also a model of Φ', i.e., $N'_I \models \Phi'$. Consequently, Φ' is entailed in N' with respect to minimal model semantics.

Both satisfiability checking and minimal model query answering are complex reasoning tasks, particularly in the context of ontologies consisting of several million formulas. The reason for this is that verifying the satisfiability of formula sets from the Bernays–Schönfinkel Horn fragment is EXPTIME complete [Pla84], and answering queries with respect to minimal model semantics is beyond standard first-order reasoning.

The next section shows related work done in the context of reasoning about ontologies.

1.3. Related work

This section depicts other work that has been done towards the development of efficient reasoning procedures for ontologies. The procedures presented in this other work can be distinguished by the expressiveness of their underlying logics, the size of axiom sets they can efficiently reason about, and the expressiveness of their query language.

The systems developed in [LPF+06, DFK+07, LTW09] extended relational databases with reasoning engines. They are suitable for reasoning about knowledge bases with

a large proportion of facts and a rather small and simple set of additional formulas. The set of additional formulas is called the *background knowledge*. These systems benefit from sophisticated techniques developed for relational databases. Because of the fact that the databases store the facts of a knowledge base, these approaches scale very well when increasing the number of facts. On the other hand, their logic for encoding the background knowledge is rather restricted. In particular, computing the transitive closure of a relation requires a complete grounding of this relation. In the case of YAGO, this is not practically feasible. This becomes even more involved when considering formulas containing transitive predicates. Such a formula may represent exponentially many ground instances.

Following [GKKS11a], answer set programming (ASP) [GL90, Bar03] has been successfully used in a variety of applications, for example, product configuration [SN98], decision support for the space shuttle [NBG+00], automatic music composition [BBVF10], automatic synthesis of multiprocessor systems [IMB+09], and inconsistency detection in large biological networks [GSTV11]. First, in the ASP approach, a given set of formulas is completely ground instantiated before applying the actual solver. Although efficient grounding tools like *lparser* [Syr98] and *gringo* [GST07, GKKS11b] have been developed, a complete grounding of the YAGO ontology involving 2 million constants and 10 million clauses with transitive predicates cause a blow up of the search space, which is outside of the scope of these grounding procedures [GKKS11a]. For example, gringo, applied to YAGO++, an extension of YAGO presented in Chapter 8, ran out of memory on a computer with 96 GB of main memory. During this execution, it generated over one billion ground instances.

Description logics (DL) [BCM+03] are widely used to encode ontologies. They are mostly decidable subsets of first-order logic, and a DL ontology usually consists of two parts, an *ABox* and a *TBox*. The ABox contains the facts of the ontology and the TBox the terminological part. The TBox contains the formulas composing the background knowledge. The computation of the subsumption hierarchy of an ontology is an essential reasoning task in description logics. Computing the subsumption hierarchy is deciding the entailment of concepts in the TBox. For example, the concept 'human' is entailed by the concept 'mammal'. The DL reasoners [SPG+07, TH06, HM01, KKS11] have been developed in order to compute the subsumption hierarchy. Experiments in [MS06] show that they are not particularly suited for answering queries in ontologies with a large ABox. In order to also efficiently answer queries in large ontologies represented in description logics, [CGL09, HMS07] have developed translations from description logics to datalog. [Sch99] shows that the well-known description logic $\mathcal{ALC}$ can be embedded into the Bernays–Schönfinkel fragment.

State of the art superposition based first-order theorem provers like E [Sch02], SPASS [WDF+09], and Vampire [RV01] are originally designed to reason about first-order theories consisting of at most several hundred complex formulas. This was motivated by automatically proving mathematical theorems and performing verification tasks on complex systems. In recent work, superposition based theorem provers have also been used to reason about knowledge bases formulated in expressive logics that are beyond standard description logics [BCM+03] and datalog [CGT89]. For example, [PSST10, HV06]

uses Vampire to reason about subsets of the SUMO [NP01b] ontology. Experiments presented in [SS11] give evidence that superposition based provers perform better than tableaux based description logic reasoners when reasoning about ontologies encoded in *OWL 2 Full* [OWL09]. The OWL 2 Full language is an expressive semantic web language that is part of the W3C standard.

The results of the last CASC competition [Sut11] show that superposition based automated theorem provers can successfully answer simple existential queries in large ontologies. The systems [Sch02, RV01, Kor08] participating in the large theory category of the CASC competition use an axiom selection heuristic [HV11] which ignores certain axioms. In a preprocessing step, this heuristic selects a small set of axioms in a goal oriented manner from the knowledge base trying to identify the axioms relevant for proving the query. The resulting small set of clauses is then processed by the respective systems. In [RRG05] an inference system and the respective heuristics have been particularly designed in order to reason about the CYC [Len95] ontology. The approach presented in [SSW+09] tries to identify relevant axioms in a database on-the-fly during the reasoning process of SPASS [WDF+09].

The LogAnswer [FGHP08] system is a natural language query answering tool which combines text analysis approaches with theorem proving techniques. The system contains a snapshot of the German Wikipedia. In order to answer a natural language query, it tries to identify text passages in the available documents that may contain the answer. After that, the query and the text passages are translated into a logical representation. This logical representation, together with some general background knowledge, is processed by the tableaux based theorem prover E-KRHyper [PW07]. If E-KRHyper finds a proof, then this proof is further processed in order to generate an answer to the query. In summary, the LogAnswer system only considers a small part of the available knowledge and is, therefore, also not complete.

All of the above mentioned heuristics are based on the assumption that the proof for an answer relies on only a small part of the knowledge base. In fact, they ignore most axioms of the knowledge base. The resulting incompleteness has two consequences. First, verifying the consistency of a knowledge base is beyond the capabilities of this heuristic. It is arguable what an answer to a query in an inconsistent ontology means. As a matter of fact, the versions of the ontologies SUMO and CYC, as used in the last CASC competition, were inconsistent. The experiments in Chapter 8 show that the procedures I have developed in this thesis prove these inconsistencies. Second, this heuristic is only capable of answering simple existential queries and only in standard first-order model semantics. Consequently, they are unable to answer queries in terms of the minimal model because already fixing the domain leads to reasoning tasks beyond standard first-order reasoning [HW10].

As a result, I was not able to find an already existing approach that covers all requirements; none of the existing approaches are at the same time sound, complete, terminating, practically feasible, and provide a query answering procedure that answers complex queries with respect to minimal model semantics.

1.4. Contribution and Structure

In this thesis, I present the first superposition based reasoning procedures that efficiently decide the satisfiability of ontologies consisting of several million axioms. The developed procedures also perform efficient entailment checks for complex formulas with respect to the minimal model semantics of the ontology. This provides a powerful query answering procedure for complex queries containing arbitrary quantifier alternations.

First, I have defined the ontology language BSH-Y2, which is a decidable fragment of first-order logic. The language BSH-Y2 is able to appropriately represent large parts of the knowledge of the widely recognized ontologies YAGO [SKW07, SKW08], SUMO [NP01a], and CYC [Len95].

Second, I have developed decision procedures for this language together with the necessary data structures. These procedures decide the satisfiability of the BSH-Y2 subsets of the above ontologies and typically answer queries in the range of a few seconds.

1.4.1. The Ontology Language BSH-Y2

The BSH-Y2 is able to encode the complete core knowledge of YAGO (10m formulas), about 90% of SUMO (83k formulas), and about 30% of CYC (1m formulas). I consider the versions of SUMO and CYC as contained in the benchmark set for automated theorem provers TPTP [Sut10].

The language BSH-Y2 is a subset of the the Bernays–Schönfinkel Horn fragment with equality and contains the following type of closed formulas

$P(a_1, \ldots, a_n)$	Ground Fact
$\forall x(S_1(x) \to S_2(x))$	Subsort Relation
$\forall x, y, z(R(x, y) \wedge R(x, z) \to y \approx z)$	Functionality
$\forall x, y, z(R(x, y) \wedge R(y, z) \to R(x, z))$	Transitivity
$\forall \overline{x}(\neg P_1(t_{11}, \ldots, t_{1n_1}) \vee \ldots \vee \neg P_k(t_{k1}, \ldots, t_{kn_k}))$	Constraints
$\forall \overline{x}(P_1(t_{11}, \ldots, t_{1n_1}) \wedge \ldots \wedge P_k(t_{k1}, \ldots, t_{kn_k}) \to P(s_1, \ldots, s_m))$	Defined Relations

where a_i are constants, x, y, z are variables, each t_{ij}, s_i is either a constant or a variable, and $\overline{x}$ denotes a sequence of variables. All of these formulas are closed, i.e., all variables are bound. The symbol R denotes a binary relation, S_i denotes a sort, and P, P_i denotes arbitrary relations. The constants represent individuals of the ontology and, consequently, are assumed to be different (unique name assumption). The components of a formula having the form $P(s_1, \ldots, s_n)$, $R(s_1, \ldots, s_n)$, and $S_i(x)$ are called atoms. The defined relations are acyclic and range restricted. This means that all variables of $P(s_1, \ldots, s_m)$ also occur in

$$P_1(t_{11}, \ldots, t_{1n_1}) \wedge \ldots \wedge P_k(t_{k1}, \ldots, t_{kn_k}).$$

The language BSH-Y2 is defined in detail in Chapter 4. The following ground fact says that Albert Einstein was born in Ulm:

$$\text{bornIn}(\text{AlbertEinstein}, \text{Ulm}). \tag{1.5}$$

The subsort relation

$$\forall x(\text{human}(x) \rightarrow \text{mamal}(x)) \tag{1.6}$$

expresses that every human is a mammal. The functionality constraint

$$\forall x, y, z(\text{bornIn}(x, y) \wedge \text{bornIn}(x, z) \rightarrow y \approx z)$$

says that bornIn is a functional relation.

$$\forall x, y, z(\text{locatedIn}(x, y) \wedge \text{locatedIn}(y, z) \rightarrow \text{locatedIn}(x, z))$$

states that the relation locatedIn is transitive. The constraint

$$\forall x, y(\neg\, \text{bornIn}(x, y) \vee \neg\, \text{bornIn}(y, x))$$

expresses that the relation bornIn is not symmetric. Finally, the defined relation

$$\forall x, y, z(\text{male}(x) \wedge \text{hasChild}(x, y) \rightarrow \text{fatherOf}(x, y))$$

defines the new relation fatherOf.

The ontologies SUMO and CYC, as contained in the TPTP, are already represented as a set of first-order formulas. I extracted from these representations the subsets contained in BSH-Y2, which I call *SUMO-Y2* and *CYC-Y2*.

Because YAGO is represented in a flat text file format, and because there was no first-order representation before now, I have developed an automatic translator from the YAGO format into a set of formulas from BSH-Y2 (Chapter 3). In this way, the core knowledge of YAGO can be translated. In addition to its core knowledge, YAGO contains meta facts, which give additional information for each fact, e.g., the time of its extraction and the respective source. I ignore the meta facts for reasoning and, therefore, I have only translated the core of YAGO. The translated core contains 10 million formulas, over 2 million individuals, and the following types of formulas: ground facts, subsort relations, functionality axioms, and transitivity axioms.

In order to also evaluate the other constructs of BSH-Y2 in the context of YAGO, I have manually added constraints and defined relations to the representation of YAGO. This extension is called YAGO++ which is presented in detail in Chapter 8. The YAGO++ ontology will be contained in the next releases of the TPTP.

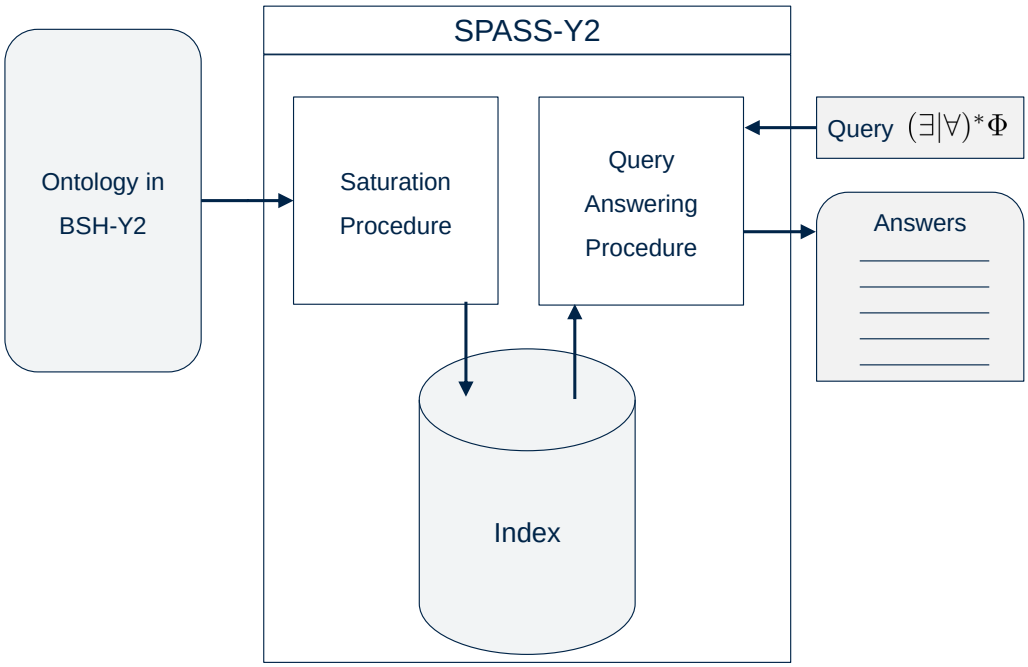

Figure 1.3.: SPASS–Y2 architecture

1.4.2. The Ontology Reasoning Engine Spass–Y2

In this thesis, I have created a new reasoning engine for large ontologies, called SPASS–Y2, which is based on the automated theorem prover SPASS [WDF+09]. Figure 1.3 gives an overview of the architecture of SPASS–Y2 and shows the components I have developed; the *saturation procedure*, the *query answering procedure*, and the *index*. The index is the underlying key data structure for an efficient implementation of both the saturation and query answering procedure. The saturation procedure decides the satisfiability of a given BSH-Y2 ontology and computes a compact representation of its minimal model. The query answering procedure performs efficiently by taking advantage of this compact minimal model representation.

Index

For an efficient implementation of both the saturation procedure and the query answering procedure, I have invented a new index for storing and accessing formulas, called *filtered context tree index* [SWW10]. In Chapter 5, I present the filtered context tree index. The filtered context tree index implements an efficient filtering technique that reduces the search space of the retrieval operations.

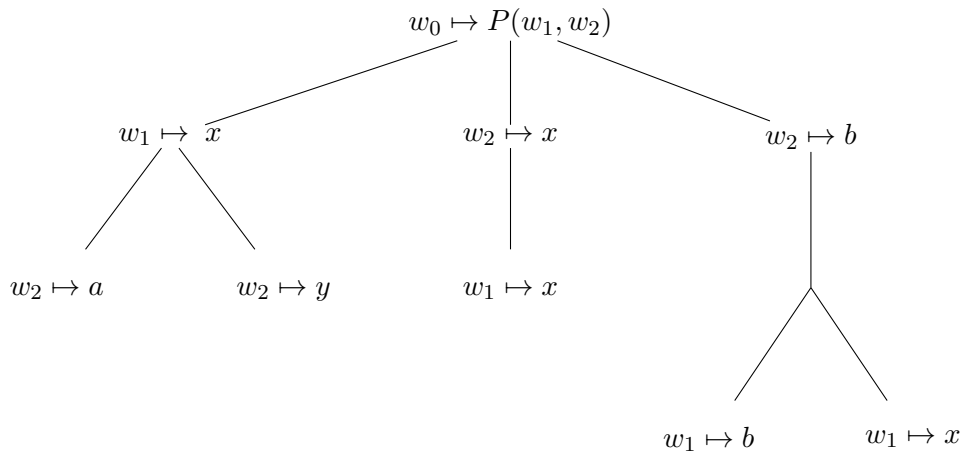

Figure 1.4.: Context tree

An index [OL80] is a data structure for storing formulas, which provides efficient retrieval procedures. For example, retrieval operations for an index data structure are the search for unifiable formulas or instances of formulas. The development of sophisticated term indexing techniques has been pivotal for successful automated theorem proving [Gra96, RSV01]. In the case of ontologies possibly containing several million formulas, this becomes even more involved. The index is a central part of the reasoning engine because it is queried several thousand times during the application of a single reasoning loop.

The filtered context tree index is based on the context tree [GNN01] index. A context tree is a tree, which stores atoms. Figure 1.4 shows a context tree. Each node of the context tree contains a substitution, and a path from the root to a leaf represents a stored atom. For example, the leftmost path represents the atom $P(x, a)$. Starting at the root and performing a retrieval operation, the filtered context tree algorithm checks whether each child, on the path, matches with the current retrieval operation. For example, the first two children of the root are unifiable with the atom $P(a, a)$ and with the substitution $x \mapsto a$, but not the last child.

In the case of a huge ontology like YAGO, a node in a context tree may contain several million children. For example, consider the retrieval for unifiable atoms in the context tree. During a retrieval operation, it is not feasible to search through all children in order to find a child that is unifiable with the query. For this reason, I have developed a filter based on the symbols of each substitution. In particular, the filtered context tree index implements a mapping mechanism. For a given retrieval atom A and a node N, the mapping returns the set of children of N that are likely to be unifiable with A. In other words, the filtering mechanism removes children of N that are not unifiable with A from the search space. As a result, it removes whole subtrees from the search space of the retrieval operation. This has been the key for an efficient implementation of both the saturation and query answering procedure.

The index that is implemented in SPASS is substitution tree indexing [Gra96]. The sub-

stitution tree index is an instance of the context tree index. For this reason, SPASS–Y2 contains the new filtering technique integrated into the implementation of the substitution tree index of SPASS. The resulting implementation of the term indexing is efficient for practical reasoning in huge ontologies. Before integrating the new filtering technique, SPASS was already unable to load the YAGO ontology within reasonable time.

Saturation Procedure

The saturation procedure of SPASS–Y2 (Chapter 6) I have developed is the first superposition based reasoning procedure that can decide the satisfiability of ontologies containing several million formulas from BSH-Y2. In addition, if the ontology is consistent, the saturation procedure returns a compact representation of its minimal model. The query answering procedure is efficient, sound, and complete in terms of this compact representation. In general, verifying the satisfiability in the Bernays–Schönfinkel Horn fragment is EXPTIME complete [Pla84]. Consequently, reasoning in an ontology with several million clauses is a hard reasoning problem.

Therefore, I have designed a reasoning procedure in such a way that the expensive reasoning applies only to a rather small part of the actual reasoning problem. I have accomplished this with the design of a two-layered reasoning procedure. This procedure separates the reasoning about transitivity from the non-transitive reasoning. For each of these layers I have designed a separate reasoning calculus. The non-transitive reasoning calculus performs a Hyperresolution reasoning, and the transitive reasoning calculus is based on an instance of the chaining calculus [BG98].

The new two-layered reasoning procedure is sound, complete, and terminating. As the experiments of Chapter 8 confirm, the new procedure is also feasible for practical reasoning in the huge ontologies YAGO++, SUMO-Y2, and CYC-Y2. It saturates the YAGO++ ontology in 16 minutes, SUMO-Y2 in 53 minutes, and it finds inconsistencies in CYC-Y2 within one minute.

Query Answering Procedure

In Chapter 7 of this thesis, I introduce a sound and complete query answering procedure that answers queries in BSH-Y2 ontologies with respect to minimal model semantics. The supported query language is a subset of first-order logic, which can express complex formulas containing arbitrary quantifier alternations. Answering queries of this query language with respect to minimal model semantics is beyond standard first-order reasoning procedures [HW10].

The query answering procedure is based on a finite domain quantifier elimination algorithm. A finite domain quantifier elimination that performs a complete instantiation is bounded by the complexity $\mathcal{O}(m^n)$ where m is the number of occurring constants and n the number of quantifiers of the query. An ontology like YAGO contains more than

two million constants. Therefore, my procedure restricts the number of the considered query instances by exploiting the compact representation of the minimal model as a saturated set of formulas. This compact representation enables the procedure to efficiently perform intermediate queries to the minimal model during the elimination of variables.

This yields a procedure which answers complex queries containing arbitrary many quantifier alternations in YAGO++ with minimal model semantics. It answers all queries of Figure 1.2; usually within a few seconds.

1.4.3. Future work

Chapter 9 describes several directions for further investigations into the next generation of search engines. These directions include extending the reasoning procedures that support more expressive ontology languages. Reasoning about time and more specific locational knowledge is a possible extension. The new version of YAGO, called YAGO 2, already contains this additional knowledge [HSB+11].

A further direction of research is the development of parallel reasoning procedures that lift the presented approaches to an Internet scale reasoning engine.

A drawback of the presented approach is the absence of a natural language interface that opens the query answering procedure also to non-expert users.

1.4.4. Summary of Achievements

This thesis presents the first superposition based reasoning procedures that efficiently decide the satisfiability of ontologies consisting of several million axioms from BSH-Y2. The set BSH-Y2 is a subset of the Bernays–Schönfinkel Horn fragment with equality. It is able to represent the YAGO ontology as well as large parts of the ontologies SUMO (SUMO-Y2) and CYC (CYC-Y2). Verifying the satisfiability of formula sets from the Bernays–Schönfinkel Horn fragment is EXPTIME complete.

Further, I have developed a query answering procedure that answers complex queries containing arbitrary quantifier alternations with respect to minimal model semantics. In general, minimal model reasoning is beyond standard first-order reasoning procedures.

In order to obtain efficient implementations of the reasoning procedures, I have developed a new index called *filtered context tree index*. This index implements an efficient filtering technique based on the symbols of an ontology. This filter provides more effective retrieval operations. For example, the index storing YAGO contains about 10 million formulas, and a retrieval operation is performed several thousand times during one reasoning loop. Therefore, efficient retrieval operation are the key for practical reasoning about huge formula sets.

I have implemented the new procedures and the new index in SPASS. The resulting new version is called SPASS–Y2. This new version is useful to practically reason about huge ontologies. It verifies the consistency of YAGO within 16 minutes, the consistency of SUMO-Y2 in 53 minutes, and it finds inconsistencies of CYC-Y2 within one minute. Answering times for complex queries with quantifier alternations are usually within a few seconds.

2. Foundations

This chapter recalls basic definitions and notions of first-order logic and first-order theorem proving, in general. It serves as the foundation upon which the work presented in this thesis relies. The notions and definitions of this chapter are mainly from [Wei01] and [BG01]. In particular, you find the syntax and semantics of standard first-order logic in Section 2.1 and an overview over the superposition based first-order reasoning framework in Section 2.2. The implementations of superposition based automated theorem provers like SPASS [WDF+09] rest upon this framework.

2.1. First-Order Logic

2.1.1. Syntax

Definition 1 (Signature)**.** *A first-order language is constructed over a signature* $\Sigma = (\mathcal{F}, \mathcal{R})$*. Assume* $\mathcal{F}$ *and* $\mathcal{R}$ *are non-empty, disjoint and finite sets. The set* $\mathcal{F}$ *is a set of function symbols and* $\mathcal{R}$ *a set of predicate symbols. In addition, assume a function* $\text{arity} : \mathcal{F} \cup \mathcal{R} \to \mathbb{N}$ *that assigns to each function symbol and predicate symbol an arity. Assume there is the equality predicate in the signature, i.e.* $\approx \in \mathcal{R}$*.*

Definition 2 (Variables)**.** *In addition to the signature* Σ *assume three countable infinite and pairwise disjoint sets of variables* $\mathcal{X}$*,* $\mathcal{U}$ *and* $\mathcal{W}$*.*

Note, the variable set $\mathcal{X}$ is the set of variables used for standard first-order terms in Definition 3. The other two sets of variables are used in the context of context tree term indexing (Section 2.2.5) and filtered context tree term indexing (Chapter 5). The set $\mathcal{U}$ is a set of *function symbol variables*. The respective definition of terms that also contain these variables is given in Definition 4. The set $\mathcal{W}$ contains variables that are used internally in term indexing data structure and the respective retrieval algorithms. They are called *index variables*. In order to distinguish these three concepts, three different sets of variables are assumed.

Definition 3 (Terms)**.** *The set of terms* $\mathcal{T}(\mathcal{F}, \mathcal{X})$ *over a signature* Σ *is recursively defined:* $\mathcal{X} \subseteq \mathcal{T}(\mathcal{F}, \mathcal{X})$ *and for every function symbol* $c \in \mathcal{F}$ *with* $\text{arity}(c) = 0$ *(*a constant*)* $c \in \mathcal{T}(\mathcal{F}, \mathcal{X})$*. For every function symbol* $f \in \mathcal{F}$ *with* $\text{arity}(f) = n$ *and* $t_1, \ldots t_n \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ *also* $f(t_1, \ldots t_n) \in \mathcal{T}(\mathcal{F}, \mathcal{X})$*. Let* $\text{vars}(t)$ *for a term* $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ *be the set of all variables occurring in* t*. If* $\text{vars}(t) = \emptyset$ *then* t *is called a ground term.*

Definition 4. *The set of terms $\mathcal{T}(\mathcal{F} \cup \mathcal{U}, \mathcal{X})$ is a superset of $\mathcal{T}(\mathcal{F}, \mathcal{X})$, i.e. $\mathcal{T}(\mathcal{F}, \mathcal{X}) \subseteq \mathcal{T}(\mathcal{F} \cup \mathcal{U}, \mathcal{X})$. Further, assume each function symbol variable $F \in \mathcal{U}$ has an associated arity,* $\text{arity}(F) = n$ *with $n > 0$. So, if $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F} \cup \mathcal{U}, \mathcal{X})$ then $F(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{F} \cup \mathcal{U}, \mathcal{X})$.*

Definition 5 (Term position)**.** *A term position ω is a word over the natural numbers. Let $t = f(t_1, \ldots, t_n)$ be a term then the set of positions* $\text{pos}(t)$ *of the term t contains the following words. The empty word $\omega = \epsilon$ is in* $\text{pos}(t)$ *and $t|_\epsilon = t$. If $t|_\omega = f(t_1, \ldots, t_n)$ then $\omega.i \in \text{pos}(t)$ and $t|_{\omega.i} = t_i$ for $i \in \{1, \ldots, n\}$. An alternative notation for $t|_\omega = s$ is $t[s]_\omega$.*

Definition 6 (Atoms)**.** *Let $\Sigma = (\mathcal{F}, \mathcal{R})$ be a signature, $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ and $P \in \mathcal{R}$ is a predicate symbol with* $\text{arity}(P) = n$ *then $P(t_1, \ldots, t_n)$ is an atom over the signature Σ. The variables of an atom are defined as* $\text{vars}(P(t_1, \ldots, t_n)) = \bigcup_i \text{vars}(t_i)$ *and* $\text{top}(P(t_1, \ldots, t_n)) = P$. *Atoms involving the equality predicate $\approx$ are written in infix notation, i.e. $t_1 \approx t_2$ with $t_1, t_2 \in \mathcal{T}(\mathcal{F}, \mathcal{X})$.*

Definition 7 (Formula)**.** *Let Σ be a signature then the language $\mathcal{L}_\Sigma$ of first-order formulas is inductively defined in terms of atoms over Σ and the logical symbols $\bot$, $\top$, $\neg$, $\wedge$, $\vee$, $\forall$ and $\exists$ as follows*

- $\bot, \top \in \mathcal{L}_\Sigma$
- $A \in \mathcal{L}_\Sigma$ *for all atoms A over Σ*
- $\neg A \in \mathcal{L}_\Sigma$ *for all atoms A over Σ*
- $\Phi_1 \wedge \Phi_2 \in \mathcal{L}_\Sigma$ *for $\Phi_1, \Phi_2 \in \mathcal{L}_\Sigma$*
- $\Phi_1 \vee \Phi_2 \in \mathcal{L}_\Sigma$ *for $\Phi_1, \Phi_2 \in \mathcal{L}_\Sigma$*
- $\forall x\ \Phi \in \mathcal{L}_\Sigma$ *for $\Phi \in \mathcal{L}_\Sigma$ and $x \in \mathcal{X}$*
- $\exists x\ \Phi \in \mathcal{L}_\Sigma$ *for $\Phi \in \mathcal{L}_\Sigma$ and $x \in \mathcal{X}$*

Note, that I use the quantifiers $\exists$ and $\forall$ in different contexts throughout this thesis, too.

Definition 8 (Sentence)**.** *Let Φ be a formula of the following form $\Phi = \forall x\ \Phi'$ or $\Phi = \exists x\ \Phi'$. In this case the occurrence of the variable x in Φ' is called bound by the quantifier $\forall$ and $\exists$, respectively. If a variable is not bound by a quantifier then it is called free. A first-order formula which has no free variables is called a sentence.*

Definition 9 (Bernays–Schönfinkel fragment)**.** *The Bernays–Schönfinkel fragment consists of formulas of the form $\exists^* \forall^* \Phi$ that do not contain any function symbols, i.e., for all $f \in \mathcal{F}$ that occur in the formula Φ it holds that* $\text{arity}(f) = 0$.

Definition 10 (Literals and Clauses)**.** *A literal is an atom or a negated atom. A clause is a disjunction of literals. If $L_1, \ldots, L_n$ are literals then $L_1 \vee \cdots \vee L_n$ is a clause. Clauses are also written in implication form $\Gamma \to \Delta$ where Γ and Δ are multisets of atoms. The*

set Γ *is interpreted as the conjunction of its elements and* Δ *as the disjunction of its elements. In addition, variables of a clause are assumed to be universally quantified, i.e., bound with the quantifier* $\forall$*. The multiset* Γ *is called the antecedent and the multiset* Δ *the succedent of the clause* $\Gamma \to \Delta$*. The empty clause* $\Gamma = \Delta = \emptyset$ *is denoted by* $\square$*. The set of predicate symbols of a clause* $C = L_1 \vee \cdots \vee L_n$ *is defined as* $\text{preds}(C) = \bigcup_i \text{top}(L_i)$*.*

Note that a clause is a sentence because all variables are assumed to be bound by a universal quantifier. For example, the clause $\neg A_1 \vee \neg A_n \vee B_1 \vee \ldots B_m$ looks as follows in implication notation $A_1, \ldots, A_n \to B_1, \ldots, B_m$ where $A_1, \ldots, A_n, B_1, \ldots B_m$ are atoms. Note, I use x, y, z to denote variables, a, b and c to denote constants, s, t, l, r to denote terms, A, B to denote atoms, L to denote literals, C, D to denote clauses and N to denote a set of clauses.

Definition 11 (Horn clauses)**.** *A clause* C *is called Horn iff it has either the form* $\Gamma \to$ *or* $\Gamma \to A$*.*

In this thesis I, consider the sort reasoning calculus [GMW97, Wei01] which treats monadic (unary) predicates independently from all other predicates. The following shows the respective definitions.

Definition 12 (Sort)**.** *A monadic predicate* S*, i.e.,* $S \in \mathcal{R}$ *and* $\text{arity}(S) = 1$*, is called a sort. The atom* $S(t)$ *for* $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ *is a sort atom.*

Definition 13 (Sorted clauses)**.** *A clause can also be written in the form* $\Theta \,\|\, \Gamma \to \Delta$*. The multiset* Θ *is called the sort constraint and solely contains sort atoms interpreted as conjunction. The multisets of atoms* Γ *and* Δ *are interpreted as in Definition 10.*

Definition 14 (Solved sort constraint)**.** *A sort constraint* Θ *is solved in a clause* $C = \Theta \,\|\, \Gamma \to \Delta$ *if for all* $S(t) \in \Theta$ *the term* t *is a variable and* $t \in \text{vars}(\Gamma \cup \Delta)$*. A sort constraint is called unsolved otherwise.*

Definition 15 (Static sort theory [GMW97])**.** *The static sort theory of a clause set* N *is the set of clauses* $\Theta \,\|\, \to S(t)$ *such that there is a clause* $\Theta' \,\|\, \Gamma \to \Delta, S(t) \in N$ *with (i)* Θ' *is solved and (ii)* Θ *is a maximal subset of* Θ' *with* $\text{vars}(\Theta) \subseteq \text{vars}(S(t))$*. The static sort theory of a clause set* N *is donated as* S_N*.*

Note, in general, the static sort theory S_N of a clause set N safely approximates the clauses of N containing positive monadic atoms [Wei01]. More precisely, $\text{S}_N \not\models \exists x_1, \ldots, x_n\ \Theta$ implies $N \not\models \exists x_1, \ldots, x_n\ \Theta$ with $\text{vars}(\Theta) = \{x_1, \ldots, x_n\}$.

In addition, I also consider the chaining calculus [BG98] which is a particular calculus efficiently reasoning about clause sets involving transitivity. The following defines transitivity axioms and transitive predicates.

Definition 16 (Transitivity axiom)**.** *A clause of the following form is called the* transitivity axiom *for the predicate* P *with* $P \in \mathcal{R}$

$$P(x, y), P(y, z) \to P(x, z)$$

Definition 17 (Transitive predicate)**.** *The chaining calculus is defined in terms of a clause set N not containing any transitivity axioms and a set of predicates* $\mathrm{Tr} \subset \mathcal{R}$ *denoting the predicates assumed to be transitive in N. The predicates of N occurring in* Tr *are called the transitive predicates of N.*

Definition 18 (Transitive theory)**.** *Let* Tr *be a set of predicate symbols from $\mathcal{R}$ then the transitive theory of* Tr *is defined as the set of transitivity axioms as follows*

$$\mathcal{A}_{\mathrm{Tr}} = \{Q(x,y), Q(y,z) \to Q(x,z) \mid Q \in \mathrm{Tr}\}$$

A clause set N not containing any transitivity axioms together with the set Tr *represent the clause set $N \cup \mathcal{A}_{\mathrm{Tr}}$.*

Definition 19 (Substitution)**.** *A substitution $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}, \mathcal{X})$ is a mapping from the set of variables into the set of terms such that $x\sigma \neq x$ for only finitely many $x \in \mathcal{V}$. The domain of a substitution σ is defined as $\mathrm{dom}(\sigma) = \{x \mid x\sigma \neq x\}$ and the codomain is defined as $\mathrm{cod}(\sigma) = \{x\sigma \mid x\sigma \neq x\}$.*

A substitution uniquely assigns the term $x\sigma$ to each variables $x \in \mathrm{dom}(\sigma)$. Consequently, the substitution σ can also be written in the form $\sigma = \{x_1 \mapsto x_1\sigma, \ldots, x_n \mapsto x_n\sigma\}$ for $\{x_1, \ldots, x_n\} = \mathrm{dom}(\sigma)$.

The composition of two substitutions $\sigma \circ \tau$ applied to a variable x is defined as the substitution $(x\sigma)\tau$. This can be simply written as $x\sigma\tau$ by omitting the brackets. A substitution σ can be lifted to a substitution over terms $\mathcal{T}(\mathcal{F}, \mathcal{X})$ as follows:

for $t = x \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ we have $t\sigma = x\sigma$

if $t = f(t_1, \ldots, t_n)$ then $t\sigma = f(t_1\sigma, \ldots, t_n\sigma)$

Likewise, if σ is a substitution then

- $P(t_1, \ldots, t_n)\sigma = P(t_1\sigma, \ldots, t_n\sigma)$
- $(\neg P(t_1, \ldots, t_n))\sigma = \neg P(t_1\sigma, \ldots, t_n\sigma)$.
- $\{A_1, \ldots, A_n\}\sigma = \{A_1\sigma, \ldots, A_n\sigma\}$.
- $(\Gamma \to \Delta)\sigma = \Gamma\sigma \to \Delta\sigma$.

Definition 20 (Unifier, Generalization, Instances)**.** *Given two terms s,t, a substitution σ is called a* unifier *if $s\sigma = t\sigma$ and* most general unifier *(*mgu*) if, in addition, for any other unifier τ of s and t there exists a substitution λ with $\sigma\lambda = \tau$. A substitution σ is called a* matcher *from s to t if $s\sigma = t$. The term s is then called a* generalization *of t and t an* instance *of s. If $s\sigma$ is ground then σ is called a grounding substitution for s or, alternatively, $s\sigma$ is a ground instance of s.*

2.1.2. Semantics

Definition 21 (Herbrand Interpretation)**.** *A* Herbrand interpretation *I over the signature Σ is a set of ground atoms over Σ. Each ground atom A is called true in I if $A \in I$. It is called false in I if $A \notin I$. The logical constant $\top$ is true in I and $\bot$ is false in I. For ground formulas Φ_1 and Φ_2 the logical connectives are interpreted in the usual way: A negated atom $\neg\Phi$ is true in I if $\Phi \notin I$. A conjunction $\Phi_1 \wedge \Phi_2$ is true in I if both Φ_1 and Φ_2 are true in I; the disjunction $\Phi_1 \vee \Phi_2$ is true in I if at least one of Φ_1 and Φ_2 is true in I; a universally quantified formula $\forall x\ \Phi$ is true in I if $\Phi\sigma$ is true in I for all substitutions σ that assign x to some ground term; an existential formula $\exists x\ \Phi$ is true in I if $\Phi\sigma$ is true in I for some substitutions σ that assign x to some ground term. A ground clause C is called true in I if one of its literals is true in I.*

Definition 22 (Model)**.** *A Herbrand interpretation I is called a model of a formula Φ iff Φ is true in I, written $I \models \Phi$. The notation $\Phi_1 \models \Phi_2$ denotes that Φ_2 is true in a Herbrand interpretation I whenever Φ_1 is true in I; alternatively $I \models \Phi_2$ whenever $I \models \Phi_1$. For clauses $C_1, \ldots C_n$ and D we write $C_1, \ldots, C_n \models D$ iff for all Herbrand interpretations I whenever $I \models C_1, \ldots, C_n$ then also $I \models D$. If N is a set of clauses then I is a model of N, written $I \models N$ iff $I \models C$ for all C in N.*

Definition 23. *For a non-ground clause C and a Herbrand interpretation I we write $I \models C$ iff for all grounding substitutions σ we have that $I \models C\sigma$.*

Definition 24 (Satisfiability)**.** *We call a set of clauses (formulas) N* satisfiable *iff there is a Herbrand interpretation I with $I \models N$. Otherwise, N is called* unsatisfiable *or inconsistent.*

Definition 25 (Theory)**.** *We call a satisfiable set of formulas (clauses) a theory.*

2.2. Superpositon based First-Order Reasoning Framework

A theorem prover can prove if a formula Φ is implied by a set of clauses N, i.e. $N \models \Phi$. A superposition based first-order theorem prover is a refutational theorem prover that proves the equivalent problem $N \cup \{\neg\Phi\} \models \bot$. Alternatively, it verifies if $N \cup \{\neg\Phi\}$ is unsatisfiable. The underlying reasoning procedure is composed of a set of inferences and reductions. The inferences span the search space by deriving new consequence. The reductions restrict the search space by replacing the clauses of the search space by simpler ones. Inferences as well as reductions are defined regarding an ordering on terms, literals and clauses. The implementation of inferences and reductions is based on a term index data structure which provides an efficient retrieval mechanism for candidate clauses that may be involved in an inference or reduction.

Based on the general superposition based first-order theorem proving framework, I have developed respective calculi and data structures that efficiently reason about huge

ontologies like YAGO. Even the query answering procedure, which lies beyond standard first-order reasoning, successfully uses this framework.

The following provides a brief overview over the first-order reasoning framework. Section 2.2.1 depicts the main reasoning loop and Section 2.2.2 shows the ordering on terms, literals and clauses which is used in the definition of the inference and reduction rules. The inferences and reductions used in this thesis are shown in Section 2.2.3 and Section 2.2.4, respectively. For a more detailed presentation of this framework consider [Wei01]. Section 2.2.5 shows the context tree term indexing and gives the definitions and algorithms from the introductory article [GNN01]. It completes this article by providing the missing retrieval algorithms.

2.2.1. First-Order Reasoning Procedure

The first-order resolution procedure considered in this work was first used in the *Otter* theorem prover [McC03, MW97, Wei01] and is the loop implemented in SPASS. It is based on a set of inferences (Section 2.2.3) and a set of reductions (Section 2.2.4). It exhaustively applies the inferences and reductions on a set of clauses N. During this process the inferences explore the search space by inferring new consequences. The reductions restrict the search space by deleting unnecessary clauses.

Algorithm 1 depicts the main loop of the superposition based first-order reasoning framework. The procedure call $\mathrm{Inf}(C, \mathrm{WorkedOff})$ performs all possible inferences between the clause C and clauses $D \in \mathrm{WorkedOff}$ and returns the set of conclusions. The procedure $\mathrm{Red}(\mathrm{Derived}, \mathrm{Usable}, \mathrm{WorkedOff})$ interreduces the three sets Derived, Usable and WorkedOff, i.e., it applies all reductions possible between the clauses of these sets.

Algorithm 1: ProofSearch

Input: Clause set N

```
1  WorkedOff := ∅;
2  Usable := N;
3  while Usable ≠ ∅ and □ ∉ Usable do
4  |   Given := C with C ∈ Usable;
5  |   Usable := Usable \{Given};
6  |   Derived := Inf(Given, WorkedOff);
7  |   WorkedOff := WorkedOff ∪{Given};
8  |   Red(Derived, Usable, WorkedOff);
9  |   Usable := Usable ∪ Derived;
10 end
11 if Usable = ∅ then
12 |   N is satisfiable
13 else if □ ∈ Usable then
14 |   N is unsatisfiable
15 end
```

In lines 4–5 a clause Given is chosen from the clause set Usable and removed from Usable. Then all inferences between Given and clauses from WorkedOff are computed in line 6 and the clause Given is inserted into WorkedOff. The clause sets Derived,Usable and WorkedOff are completely interreduced in line 8. This means each clause C of Derived is fully reduced with clauses from Usable and WorkedOff. This process is called *forward reduction*. After that, all clauses of Usable and WorkedOff are reduced with C. This is called *backward reduction*. After complete interreduction the remaining clauses in Derived are added to the set Usable in line 9. The loop repeats this process until either the empty clause has been derived from an inference or there are no clauses left in Usable. This loop ensures that all clauses are processed and all inferences between clauses are performed in the limit. If the loop terminates then the given clause set N is saturated (Definition 33). This procedure together with appropriate inferences (Section 2.2.3) and reductions (Section 2.2.4) is refutationally complete. This means that whenever the clause set N is unsatisfiable this procedure derives the empty clause $\square$ [BG01].

2.2.2. Ordering on Terms and Clauses

Superposition based calculi are defined in relation to an ordering on terms and literals. Only maximal literals in a clause are considered for performing inferences. The introduction of such an ordering restricts the number of inferences between two clauses without loosing completeness. The usual ordering on terms which is lifted to literals and clauses, is described in this section. The work of this thesis is also based on the chaining calculus [BG98] which requires an extension of the usual term ordering. This extension is called *admissible* ordering and is shown in this section, as well.

The actual orderings used for the implementation of superposition based calculi are mostly variants of the *Knuth-Bendix Ordering* [KB70] or the *recursive path ordering with status* [Der82]. These orderings are defined regarding the tree structure of terms and formulas. Assume a strict ordering $>$ on the signature symbols in Σ which is called a *precedence*. Let *weight* be a function weight : $\Sigma \to \mathbb{N}$ assigning a natural number to each signature symbol. The function weight is extended to a function weight : $\mathcal{T}(\mathcal{F}, \mathcal{X}) \to \mathbb{N}$ as follows:

- if $t \in \mathcal{X}$ then $\text{weight}(t) = k$, where $k = \min(\{\text{weight}(c) \mid c \in \mathcal{F},\ \text{arity}(c) = 0\})$
- if $t = f(t_1, \ldots, t_n)$ then $\text{weight}(t) = \text{weight}(f) + \sum_i \text{weight}(t_i)$

The number of occurrences of a term s in a term t is defined as

$$\text{occ}(s, t) = |\{p \in \text{pos}(t) \mid t|_p = s\}|$$

and *status* is a mapping *status* : $\Sigma \to \{\mathit{left}, \mathit{right}, \mathit{mul}\}$. The ordering $\succ^{lex}$ is the lexicographic extension of the strict ordering $\succ$ and is defined by $(t_1, \ldots, t_k) \succ^{lex} (s_1, \ldots, s_l)$ if $t_i \succ s_i$ for some $i \in \{1, \ldots, k\}$ and $t_j = s_j$ for all $j \in \{1, \ldots, i-1\}$. Let M, N be multisets then the ordering $\succ^{mul}$ is the multiset extension of the strict ordering $\succ$ and is defined by $M \succ^{mul} N$ if $M \neq N$ and for all $n \in N \setminus M$ there is a $m \in M \setminus N$ with $m \succ n$.

Definition 26 (KBO). *Let $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ be two terms then $t \succ_{\text{kbo}} s$ iff $\text{occ}(x, t) \geq \text{occ}(x, s)$ for all variables x with $x \in \text{vars}(t) \cup \text{vars}(s)$ and*

1. $\text{weight}(t) > \text{weight}(s)$ *or*
2. $\text{weight}(t) = \text{weight}(s)$ *and* $t = f(t_1, \ldots, f_k)$ *and* $s = g(s_1, \ldots, s_l)$ *and*
   a) $f > g$ *or*
   b) $f = g$ *and*
      i. $\mathit{status}(f) = \mathit{left}$ *and* $(t_1, \ldots, t_k) \succ^{\mathit{lex}}_{\text{kbo}} (s_1, \ldots, s_l)$ *or*
      ii. $\mathit{status}(f) = \mathit{right}$ *and* $(t_k, t_{k-1} \ldots, t_1) \succ^{\mathit{lex}}_{\text{kbo}} (s_l, s_{l-1}, \ldots, s_1)$

If the precedence $>$ is total on Σ then KBO is total on ground terms. Note, the case $\mathit{status}(f) = \mathit{mul}$ can also be defined but is not practically useful [Wei01].

Definition 27 (RPOS). *Let $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ then $t \succ_{\text{rpos}} s$ iff*

1. $t \in \text{vars}(s)$ *and* $t \neq s$ *or*
2. $t = f(t_1, \ldots, t_k)$ *and* $s = g(s_1, \ldots, s_l)$ *and*
   a) $t_i \succeq_{\text{rpos}} s$ *for some* $i \in \{1, \ldots, k\}$ *or*
   b) $f > g$ *and* $t \succ_{\text{rpos}} s_j$ *for all* $j \in \{1, \ldots, l\}$ *or*
   c) $f = g$ *and*
      i. $\mathit{status}(f) = \mathit{left}$ *and* $(t_1, \ldots, t_k) \succ^{\mathit{lex}}_{\text{rpos}} (s_1, \ldots, s_l)$ *and* $t \succ_{\text{rpos}} s_j$ *for all* $j \in \{1, \ldots, l\}$ *or*
      ii. $\mathit{status}(f) = \mathit{right}$ *and* $(t_k, t_{k-1} \ldots, t_1) \succ^{\mathit{lex}}_{\text{rpos}} (s_l, s_{l-1} \ldots, s_1)$ *and* $t \succ_{\text{rpos}} s_j$ *for all* $j \in \{1, \ldots, l\}$
      iii. $\mathit{status}(f) = \mathit{mul}$ *and* $\{t_1, \ldots, t_k\} \succ^{\mathit{mul}}_{\text{rpos}} \{s_1, \ldots, s_l\}$

If the precedence $>$ is total on Σ then RPOS is total on ground terms.

The precedence is also defined on predicate symbols which provides an extension of KBO and RPOS to atoms. Further, if the precedence is total on the symbols in Σ then KBO and RPOS are total on ground atoms.

An ordering on atoms can be extended to literals and clauses as follows.

Definition 28 (Literal/Clause ordering). *Let $\succ$ be an ordering on atoms. Clauses are considered as the multiset extension of occurrences of atoms. An atom A in the antecedent of a clause is assumed to be the multiset $\{\{A, \top\}\}$ and in the succedent the multiset $\{\{A\}, \{\top\}\}$. The constant $\top$ is always assumed to be minimal with respect to $>$. An ordering on clauses is the multiset extension of literal occurrences in a clause. The notation $\succ$ is overloaded also denoting the ordering on literals and clauses.*

In addition to this ordering, which is well-founded and total on ground terms, the chaining calculus [BG98] requires an ordering that is admissible for the transitive predicates (Definition 17) of a clause set N.

Definition 29 (Admissible Ordering)**.** *Let $\succ$ be an ordering on ground terms and literals and $max(s,t)$ be the maximum of the two terms s and t with respect to $\succ$. The ordering $\succ$ is called* admissible [BG98] *if*

- *it is well-founded and total on ground terms and literals,*
- *it is* compatible with reduction on maximal subterms*, i.e., for literals L and L', it holds that $L \succ L'$ whenever L and L' contain the same transitive predicate symbol Q, and the maximal subterm of L' is strictly smaller than the maximal subterm of L,*
- *it is* compatible with goal reduction*, i.e., for atoms A and B, it holds*
  - *$\neg A \succ A$ for all ground atoms A,*
  - *$\neg A \succ B$ whenever A is an atom $Q(s,t)$ and B is an atom $Q(s',t')$, such that Q is a transitive predicate and $max(s,t) \succeq max(s',t')$,*
  - *$\neg A \succ \neg B$ whenever A is an atom $Q(s,s)$ and B atom $Q(s,t)$ or $Q(t,s)$, where Q is a transitive predicate and $s \succ t$.*

An ordering on ground clauses is called admissible *if it is the multiset extension of an admissible ordering on literals.*

Definition 30 (Maximal literal)**.** *An atom A is called maximal in a clause $\Gamma \to \Delta$ ($\Theta \,\|\, \Gamma \to \Delta$) if there is no atom $A' \in \Gamma \cup \Delta$ such that $A' \succ A$. It is called strictly maximal if there is no $A' \in \Gamma \cup \Delta$ with $A' \succeq A$.*

Note, that Θ is not considered for maximality of literals because sort atoms are treated independently. In particular, sort predicates are assumed to have a precedence smaller than any other predicate symbol [Wei01].

Definition 31 (Reductive clause)**.** *A clause $\Gamma \to \Delta, A$ ($\Theta \,\|\, \Gamma \to \Delta, A$) is reductive for the atom A iff A is strictly maximal in the clause.*

2.2.3. Inferences

The standard first-order reasoning framework contains a variety of reasoning calculi. The current section presents only those three calculi that are used in the reasoning procedures of this thesis. These calculi are the sort reasoning calculus, the hyperresolution calculus and the chaining calculus. Each of these has specific properties. The appropriate combination and modification of these three calculi leads to a practically successful reasoning and query answering procedure for huge knowledge bases like YAGO.

Let N be a clause set. An *inference* is defined in terms of clauses $C_1, \ldots C_n \in N$ and is denoted in the following form

$$\frac{C_1 = \Gamma_1 \to \Delta_1 \quad \ldots \quad C_n = \Gamma_n \to \Delta_n}{D = \Gamma \to \Delta}$$

where the clause D is a logical consequence of the clauses $C_1, \ldots, C_n$; more formally

$$C_1, \ldots, C_n \models D$$

The clauses $C_1, \ldots C_n$ are called *premises* and D the *conclusion*. A set of inferences is called a *calculus* or an *inference system*.

Beside the ordering there exists also the selection mechanism which allows to define a reasoning strategy.

Definition 32 (Free selection)**.** *A* free selection *is a mapping that selects a set of negative atoms in each clause C of a clause set N.*

Following the notations from [Wei01], the rules use the clause notation of Definition 13 in order to make clear that sort atoms are treated independently from all other atoms.

Sort Reasoning Calculus

Let N be a set of clauses and S_N be the static sort theory of N. The sort reasoning calculus handles sort atoms independently from all other atoms of a clause set N. These calculus rules can be implemented more efficiently than the standard reasoning calculus by using particular data structures [Wei01].

Actually, the sort reasoning calculus simulates a particular ordering and selection strategy on the standard calculus [GMW97]. For all subsort declarations $S(x) \to T(x)$ of N it holds that $T \succ S$. This ordering is well-defined if the static sort theory S_N is acyclic. Whenever a clause has an unsolved constraint, this constraint is selected. Finally, all sort predicates occurring in a clause set N are assumed to be smaller than all the other predicates.

Empty sort

$$\frac{S(x), \Theta_1 \,\|\, \Gamma_1 \to \Delta_1 \qquad \Theta_2 \,\|\, \Gamma_2 \to \Delta_2, S(s)}{(\Theta_1, \Theta_2 \,\|\, \Gamma_1, \Gamma_2 \to \Delta_1, \Delta_2)\sigma},$$

$x \notin \mathrm{vars}(\Gamma_1 \cup \Delta_1)$, Θ_2 is solved, and $S(s)$ is maximal in $\Theta_2 \,\|\, \Gamma_2 \to \Delta_2, S(s)$.

Sort resolution

$$\frac{S(t), \Theta_1 \,\|\, \Gamma_1 \to \Delta_1 \quad \Theta_2 \,\|\, \Gamma_2 \to \Delta_2, S(s)}{(\Theta_1, \Theta_2 \,\|\, \Gamma_1, \Gamma_2 \to \Delta_1, \Delta_2)\sigma},$$

where σ is the most general unifier of t and s, t is not a variable, Θ_2 is solved, and $(S(s))\sigma$ is strictly maximal in $(\Theta_2 \,\|\, \Gamma_2 \to \Delta_2, S(s))\sigma$.

Resolution Calculus

Ordered resolution is a special case of standard resolution which respects the special treatment of sorts via sort reasoning as well as the ordering and selection restriction [Wei01].

Ordered resolution

$$\frac{D = \Theta' \,\|\, \Gamma' \to A \quad C = \Theta \,\|\, \Gamma, B \to \Delta}{(\Theta, \Theta' \,\|\, \Gamma, \Gamma' \to \Delta)\sigma},$$

where σ is the most general unifier of A and B, Θ and Θ' are solved, no literal in Γ' is selected, $A\sigma$ is strictly maximal in $D\sigma$, $B\sigma$ is selected or it is strictly maximal in $C\sigma$, and no literal is selected in Γ.

Ordered Hyperresolution

$$\frac{(1 \leq i \leq k) \quad \Theta_i \,\|\, \Gamma_i \to \Delta_i, A_i \quad \Theta \,\|\, B_1, \ldots, B_k, B_{k+1}, \ldots, B_n \to \Delta}{(\Theta, \Theta_1, \ldots, \Theta_n \,\|\, \Gamma_1, \ldots, \Gamma_k, B_{k+1}, \ldots, B_n \to \Delta)\sigma},$$

where $k \geq 1$, σ is the simultaneous most general unifier of A_i and B_i, Θ and all Θ_i are solved, and all $A_i\sigma$ are strictly maximal in $(\Theta_i \,\|\, \Gamma_i \to \Delta_i, A_i)\sigma$ respectively, for all $i \in \{1, \ldots k\}$.

The clause $\Theta \,\|\, B_1, \ldots, B_k, B_{k+1}, \ldots, B_n \to \Delta$ is called the *nucleus* and the clauses $\Theta_i \,\|\, \Gamma_i \to \Delta_i, A_i$ are called *electrons*.

The application of the hyperresolution rule can be simulated by several applications of resolution with a particular selection strategy and by dropping all intermediately inferred clauses [FLHT01, BG01].

This rule is a generalization of ordered hyperresolution used in the standard superposition framework. For example, the rule ordered hyperresolution [Wei01] is the instance of the above rule with $k = n$.

The reasoning calculus that I have developed in Chapter 6 uses an instance of this general ordered hyperresolution rule because it separates reasoning about transitive predicates from reasoning about non-transitive predicates. This is similar to the sort reasoning calculus which treats sort predicates separately.

Chaining Calculus

The ordered resolution and hyperesolution calculi usually do not perform very well on theories containing transitivity axioms. This results from the fact that these inferences always span the whole transitive closure during an application of Algorithm 1. If considering clause sets of the size of YAGO, effectively reasoning about transitive relations becomes crucial. Ordered chaining [BG98] provides a calculus that often reasons about

transitive theories more efficiently than the standard resolution approach and, in particular, it turned out that it is very efficient for reasoning about transitivity in the first-order representation of YAGO.

The calculus assumes a given clause set N over a signature Σ which does not contain any transitivity axioms. All the transitive predicates are contained in the set set $\mathrm{Tr} \subset \mathcal{R}$ (Definition 17). The respective transitivity theory (Definition 18) is

$$\mathcal{A}_{\mathrm{Tr}} = \{Q(x,y), Q(y,z) \to Q(x,z) \mid Q \in \mathrm{Tr}\}$$

The paper [BG98] shows that the below calculus together with ordered resolution is sound and complete in terms of the transitivity theory. More precisely, if $N \cup \mathcal{A}_{\mathrm{Tr}} \models \bot$ then the chaining calculus together with ordered resolution applied to N and Tr derives the empty clause $\Box$.

Ordered chaining

$$\frac{\Theta_1 \,\|\, \Gamma_1 \to \Delta_1, Q(l,s) \quad \Theta_2 \,\|\, \Gamma_2 \to \Delta_2, Q(t,r)}{C = \Theta_1, \Theta_2 \,\|\, \Gamma_1, \Gamma_2 \to \Delta_1, \Delta_2 Q(l,r)\sigma}$$

where $Q \in \mathrm{Tr}$, σ is the most general unifier of s and t, Θ_1 and Θ_2 are solved, no literal of Γ_1 is selected, $Q(l,s)\sigma$ is strictly maximal with respect to $\Gamma_1\sigma$ and $\Delta_1\sigma$, $Q(t,r)\sigma$ is strictly maximal with respect to $\Gamma_2\sigma$ and $\Delta_2\sigma$, $l\sigma \not\succeq s\sigma$ and $r\sigma \not\succeq t\sigma$, and nothing is selected in Γ_1 and Γ_2.

Negative chaining

$$\frac{\Theta_1 \,\|\, \Gamma_1 \to \Delta_1, Q(l,s) \quad \Theta_2 \,\|\, \Gamma_2, Q(t,r) \to \Delta_2}{(\Theta_1, \Theta_2 \,\|\, \Gamma_1, \Gamma_2, Q(s,r) \to \Delta_1, \Delta_2)\sigma}$$

where $Q \in \mathrm{Tr}$, σ is the most general unifier of l and t, Θ_1 and Θ_2 are solved, no literal of Γ_1 is selected, $Q(l,s)\sigma$ is strictly maximal with respect to $\Gamma_1\sigma$ and $\Delta_1\sigma$, $s\sigma \not\succeq l\sigma$, $r\sigma \not\succeq t\sigma$ and $Q(t,r)$ is selected or it is maximal with respect to $(\Theta_2 \,\|\, \Gamma_2, Q(t,r) \to \Delta_2)\sigma$, and no other atom is selected in Γ_2, and

$$\frac{\Theta_1 \,\|\, \Gamma_1 \to \Delta_1, Q(l,s) \quad \Theta_2 \,\|\, \Gamma_2, Q(t,r) \to \Delta_2}{(\Theta_1, \Theta_2 \,\|\, \Gamma_1, \Gamma_2, Q(t,l) \to \Delta)\sigma}$$

where $Q \in \mathrm{Tr}$, σ is the most general unifier of s and r, Θ_1 and Θ_2 are solved, no literal of Γ_1 is selected, $Q(l,s)\sigma$ is strictly maximal with respect to $\Gamma_1\sigma$ and $\Delta_1\sigma$, $l\sigma \not\succeq s\sigma$, $t\sigma \not\succ r\sigma$, $Q(t,r)$ is selected or it is maximal with respect to $(\Theta_2 \,\|\, \Gamma_2, Q(t,r) \to \Delta_2)\sigma$, and no other atom is selected in Γ_2

2.2.4. Reductions

Standard Redundancy Criterion

A clause C is *redundant* with respect to a set of clauses N if there exists clauses $C_1, \ldots, C_k \in N$ such that $C_1, \ldots C_k \models C$ and $C \succ C_j$, for all j with $1 \leq j \leq k$.

An inference π is *redundant* with respect to N if either one of its premises is redundant, or else there exists a set of clauses $C_1, \ldots, C_k \in N$ such that the conclusion of π is true in every model of $C_1, \ldots, C_k$ and $C \succ C_j$, for all j with $1 \leq j \leq k$, where C is the maximal premise of π.

Definition 33 (Saturation)**.** *A set of clauses N is* saturated (up to redundancy) *with respect to some inference system, if all inferences from N are redundant.*

Reduction rules

In contrast to the inferences, a *reduction* rule reduces the search space by deleting clauses or by reducing clauses to simpler ones. A reduction is denoted as

$$\mathcal{R}\frac{C_1 \quad \ldots \quad C_n}{\begin{array}{c} D_1 \\ \vdots \\ D_m \end{array}} \tag{2.1}$$

where the clause above the bar $C_1, \ldots, C_n$ from N are replaced in N by the clauses below the bar $D_1, \ldots, D_m$.

The reductions implement special redundancy criteria which are used in the superposition based first-order reasoning framework. In this thesis, I consider the below depicted reduction rules.

Tautology Deletion

$$\mathcal{R}\frac{\Theta \,\|\, \Gamma \to \Delta}{}$$

where $\models \Theta \,\|\, \Gamma \to \Delta$.

Definition 34 (Subsumption)**.** *A clause $\Theta_1 \,\|\, \Gamma_1 \to \Delta_1$ subsumes a clause $\Theta_2 \,\|\, \Gamma_2 \to \Delta_2$ if there is a matcher σ such that $\Theta_1\sigma \subseteq \Theta_2$, $\Gamma_1\sigma \subseteq \Gamma_2$ and $\Delta_1\sigma \subseteq \Delta_2$.*

Subsumption Deletion

$$\mathcal{R}\frac{\Theta_1 \,\|\, \Gamma_1 \to \Delta_1 \quad \Theta_2 \,\|\, \Gamma_2 \to \Delta_2}{\Theta_1 \,\|\, \Gamma_1 \to \Delta_1}$$

where $\Theta_1 \,\|\, \Gamma_1 \to \Delta_1$ subsumes $\Theta_2 \,\|\, \Gamma_2 \to \Delta_2$.

The subsumption check is NP–complete in general. Today's theorem provers use an approximation of the general definition which can be efficiently decided. For an overview consider [Wei01].

Reductions for Sort Reasoning

Sort simplification

$$\mathcal{R}\frac{S(t), \Theta \,\|\, \Gamma \to \Delta}{\Theta \,\|\, \Gamma \to \Delta} \ ,$$

where $\mathrm{S}_N \models \forall x_1, \ldots, x_n (S_1(x_1), \ldots, S_n(x_n) \to S(t))$, $\{S_1(x_1), \ldots, S_n(x_n)\}$ is the maximal subset of Θ with $\{x_1, \ldots, x_n\} \in \mathrm{vars}(t)$, and S_N is a sort theory from the clause set N.

Note, for an arbitrary sort theory S_N the following condition:

$$\mathrm{S}_N \models \forall x_1, \ldots, x_n (S_1(x_1), \ldots, S_n(x_n) \to S(t)),$$

can always be decided in polynomial time [Wei98].

Static Soft Typing

$$\mathcal{R}\frac{S(x), \Theta \,\|\, \Gamma \to \Delta}{} \ ,$$

if $S_N \not\models \exists x \, S(x)$.

2.2.5. Context Tree Term Indexing

Context trees are a term indexing data structure that stores and manages terms involved in processing a first-order reasoning problem. An index has the functionality of a database for automated theorem proving that particularly implements the specific retrieval operations occurring in automated theorem proving applications. They build the underlying data structure for an efficient implementation of the inference and reduction rules. In particular, they efficiently implement the retrieval for terms that are possibly involved in an application of an inference or reduction rule. The invention of term indexing data structures was the key for successful automated superposition based first-order theorem proving [OL80, RSV01, Gra96, NHRV01].

Likewise, powerful term indexing techniques are essential for reasoning in huge ontologies. Therefore, I have further advanced the context tree index [GNN01, GNN04] such that the resulting new index is also efficient for huge clause sets consisting of several million clauses. The new term index is presented in Chapter 5.

The context tree term index is a generalization of the substitution tree index [Gra96]. The following section shows the definitions of the context tree term index following notions from [Gra96] as well as the algorithms implementing the respective retrieval and maintenance operations. This section also completes the introductory article of the context tree index [GNN01] which only presents the algorithms for the retrieval of generalizations.

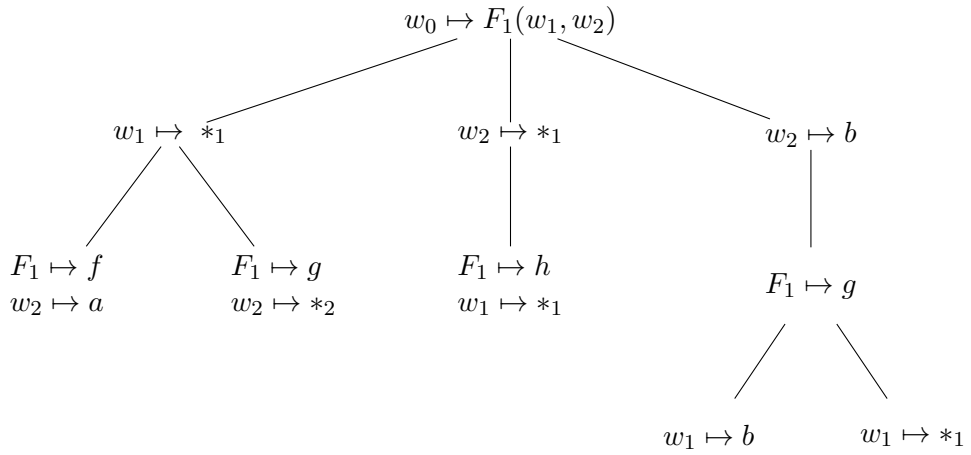

Figure 2.1.: Context tree

Context Trees

Compared to substitution trees, context trees can additionally share common subterms even if they occur below different function symbols via the introduction of extra variables for function symbols $\mathcal{U}$. These variables are called function variables. For example, the terms $f(s,t)$ and $g(s,t)$ can be represented as $F_1(s,t)$ with children $F_1 = f$ and $F_1 = g$. The function variable F_1 represents a single function symbol. In the context of deep formulas, this potentially increases the degree of sharing in a index structure. In order to increase the sharing potential, variables of a term are normalized before inserting them into the index.

Definition 35 (Normalized variables)**.** *Assume a set of variables* $\{*_1, *_2, \dots\} \subseteq \mathcal{X}$ *and a total order defined on these variables* $*_i <^* *_{i+1}$*. A normalization is a renaming substitution* σ *of the variables of a term* t *such that (i)* $\mathrm{cod}(\sigma) \subset \{*_1, *_2, \dots\}$*, (ii) for* $*_n = \max(\mathrm{cod}(\sigma))$ *it holds that* $|vars(t)| = n$ *and (iii) if* ω_1 *and* ω_2 *are the smallest positions of* t *such* $x = t|_{\omega_1}$*,* $x \in \mathrm{vars}(t)$ *and* $y = t|_{\omega_2}$*,* $y \in \mathrm{vars}(t)$ *with* $\omega_1 < \omega_2$ *then it holds that* $t\sigma|_{\omega_1} <^* t\sigma|_{\omega_2}$*.*

For example, consider the two terms $f(x,y)$ and $f(u,v)$. Both of them become the term $f(*_1, *_2)$ after the normalization; normalized variables are denoted by $*_i$. Figure 2.1 depicts a context tree containing the terms $f(*_1, a)$, $g(*_1, *_2)$, $h(*_1, *_1)$, $g(b,b)$, and $g(*_1, b)$. For context trees the notion of a substitution σ is extended to a substitution $\sigma : \mathcal{X} \cup \mathcal{U} \cup \mathcal{W} \to \mathcal{T}(\mathcal{F} \cup \mathcal{U}, \mathcal{X} \cup \mathcal{W})$.

Definition 36 (Context Tree)**.** *A context tree is a tree* $T = (V, E, \mathrm{subst}, v_r)$ *where* V *is a set of vertices,* $E \subset V \times V$ *is the edge relation, the function* subst *assigns to each vertex a substitution,* $v_r \in V$ *is the root node of* T*, and the following properties hold:*

1. *each node is either a leaf or an inner node with at least two children.*

2. *for every path* $v_1 \dots v_n$ *from the root* ($v_1 = v_r$) *to any node it holds:*

$$\mathrm{dom}(\mathrm{subst}(v_i)) \cap \bigcup_{1 \leq j < i} \mathrm{dom}(\mathrm{subst}(v_j)) = \emptyset$$

3. *for every path* $v_1 \dots v_n$ *from the root* ($v_1 = v_r$) *to a leaf* v_n

$$\mathrm{vars}(\mathrm{cod}(\mathrm{subst}(v_1) \circ \dots \circ \mathrm{subst}(v_n))) \subset \mathcal{X}$$

Each node in a context tree, which is not a leaf node, must have at least two subtrees due to the first condition. The second condition ensures that each variable is bound at most once along a path. The third condition assures that all terms represented by a path from the root to a leaf are from $\mathcal{T}(\Sigma, \mathcal{X})$.

A term that is stored in a context tree is represented by a path from the root to a leaf. The respective term can be obtained by the composition of the substitutions along this path.

Definition 37 (Variables of a path)**.** *Let* $v_1, \dots, v_n$ *be a path from the root of a context tree to a node* v_n *then the set of variables of this path is*

$$\mathrm{vars}(v_1, \dots, v_n) = \bigcup_{i \in \{1 \dots n\}} \mathrm{vars}(\mathrm{cod}(\mathrm{subst}(v_i))) \setminus \bigcup_{i \in \{1 \dots n\}} \mathrm{dom}(\mathrm{subst}(v_i))$$

Note, for a path $v_r = v_1, \dots, v_n$ of a context tree from the root v_r to a leaf v_n, it holds that $\mathrm{vars}(v_1, \dots, v_n) \subset \mathcal{X}$ because of Condition 3 of Definition 36.

Algorithms for Context Trees

This section shows the algorithms for context trees implementing the standard operations for term indexing structures. The standard operations of term indexing data structures can be separated into two categories. The first are the retrieval algorithms. These operations query the index for unifiable terms, instantiations, and generalizations of a given query term. In the second category are the algorithms for the maintenance of the indexing structure. These are the algorithms for insertion of terms into the index and deletion of terms from the index.

Retrieval Algorithm

The query algorithms for unifiable terms, instantiations, and generalizations are based on a common lookup procedure which traverses the tree and applies for the substitution of each visited node the procedure Test. The procedure Test is either the test for unifiability, the test for instantiation, or the test for generalization. If Test is successful it returns the the tuple (true, σ) where σ is the respective unifier or instance of the input.

The query that is given to the lookup function is a *query substitution* containing the query term rather than the query term itself. This means, if t is the query term, then the respective query substitution is $\{w_0 \mapsto t\}$ where w_0 is the first index variable occurring in the context tree.

Algorithm 2: Lookup

Input: context tree $T = (V, E, \text{subst}, v_r)$, $v \in V$, substitution ρ, function Test

```
1 HITS = ∅;
2 if Test(subst(v), ρ) = (true, σ) then
3 |   if isLeaf(v) then return {v};
4 |   foreach (v, v') ∈ E do
5 |   |   HITS = HITS ∪ Lookup(T, v', ρ ∘ σ, Test);
6 |   end
7 end
8 return HITS;
```

Lookup The lookup procedure Lookup (Algorithm 2) expects a context tree T, a node v, a query substitution ρ and the test function Test. The node v is initially set to the root node of T and it is the current examined node of T during the recursive application of Lookup. The substitution ρ is an accumulator argument. It is the composition (line 5) of the initial query substitution and all substitutions σ computed in line 2 during the recursive application of Lookup. The function Test is one of the functions UnifyTest (Algorithm 4), GenTest (Algorithm 6), or InstTest (Algorithm 8) which tests two substitutions for unifiability, generalization, or instantiation, respectively.

Theorem 38 (Correctness and completeness of Lookup [Gra96])**.** *Let t be a term,* Test *be one of the test functions for unification, generalization, or instantiation, $\rho = \{w_0 \mapsto t\}$. Further, let v_n be a leaf, $(v_r, v_1) \dots (v_{n-1}, v_n)$ be a path from the root v_r to v_n with $\tau_i = \text{subst}(v_i)$ then $\text{vars}(\sigma) \subset \mathcal{X} \cup \mathcal{U}$.*

$$v_n \in \text{Lookup}(T, v_r, \rho, \text{Test}) \;\Leftrightarrow\; \text{Test}(\tau_1 \circ \cdots \circ \tau_n, \rho) = (true, \sigma) \tag{2.2}$$

Unification The unification test of two substitutions τ and ρ tests if there is a substitution σ such that for all $x \in \mathrm{dom}(\tau)$ it holds $x\tau\rho\sigma = x\rho\sigma$. Note, that ρ occurs on both sides of the equation. The substitution ρ works as an accumulator argument of Lookup (Algorithm 2) and it may bind variables of $x\tau$. These bindings have to be also respected in the test function. The respective test procedure UnifyTest is depicted in Algorithm 4. The procedure UnifyTest uses the procedure TermUnify (Algorithm 3) which checks for two given terms s and t whether they are unifiable, i.e., does there exist a substitution σ with $s\sigma = t\sigma$. The correctness proof of UnifyTest for substitutions trees is given in [Gra96]. This proof can easily be extended to context trees.

Algorithm 3: TermUnify

Input: term s, term t, substitution σ

```
1  if s = x then
2  |  if sσ = t then
3  |  |  return (true, σ)
4  |  else if s ∉ dom(ρ) then
5  |  |  σ = σ ∘ {s ↦ t};
6  |  |  return (true, σ);
7  |  else
8  |  |  return (false, ∅);
9  |  end
10 else if t = x then
11 |  if s = tσ then
12 |  |  return (true, σ)
13 |  else if t ∉ dom(σ) then
14 |  |  σ = σ ∘ {t ↦ s};
15 |  |  return (true, σ);
16 |  else
17 |  |  return (false, ∅);
18 |  end
19 else if s = F(s_1, ..., s_n) and t = f(t_1, ..., t_n) then
20 |  foreach i ∈ {1, ... n} do
21 |  |  (r, σ) = TermUnify(s_i, t_i, σ);
22 |  |  if r = false then return (false, ∅);
23 |  end
24 |  if F ∈ dom(σ) ∧ Fσ ≠ f then return (false, ∅);
25 |  if Fσ = f then return (true, σ) else return (true, σ ∘ {F ↦ f});
26 end
27 return (false, ∅);
```

Algorithm 4: UnifyTest

Input: substitution τ, substitution ρ

```
1 σ = ∅;
2 foreach x ∈ dom(τ) do
3 |   (r, σ) = TermUnify(xτ,xρ,σ);
4 |   if r = false then return (false,σ)
5 end
6 return (true, σ);
```

Generalization The test function for generalization GenTest (Algorithm 6) checks for two given substitutions τ and ρ if there exists a substitution σ such that for all $x \in \mathrm{dom}(\tau) : x\tau\rho\sigma = x\rho$. Note, ρ occurs on both sides because ρ is the accumulator argument of Lookup (Algorithm 2) and may bind variables of $x\tau$. The implementation of this procedure is based on TermGen (Algorithm 5) that tests for two given terms s and t if s is a generalization of t, i.e. if a substitution σ exist with $s\sigma = t$. The correctness proof of GenTest for substitutions trees is given in [Gra96]. This proof can easily be extended to context trees.

Algorithm 5: TermGen

Input: term s, term t, substitution σ

```
1  if s = x then return (true, σ ∘ {x ↦ t});
2  if s = F(s_1, ..., s_n) and t = f(t_1, ..., t_n) then
3  |   foreach i ∈ {1, ... n} do
4  |   |   (r, σ) = TermGen(s_i, t_i, σ);
5  |   |   if r = false then return (false, ∅);
6  |   end
7  |   if F ∈ dom(σ) ∧ Fσ ≠ f then return (false, ∅);
8  |   if Fσ = f then return (true, σ) else return (true, σ ∘ {F ↦ f});
9  end
10 return (false, ∅);
```

Algorithm 6: GenTest

Input: substitution τ, substitution ρ

```
1 σ = ∅;
2 foreach x ∈ dom(τ) ∪ dom(ρ) do
3 |   (r, σ) = TermGen(xτ,xρ,σ);
4 |   if r = false then return (false,σ)
5 end
6 return (true, σ);
```

Instance The test function for instantiation InstTest (Algorithm 8) checks for two given substitutions τ and ρ if there exists a substitution σ such that for all $x \in \mathrm{dom}(\tau)$: $x\tau\rho\sigma = x\rho\sigma$ and $\mathrm{dom}(\sigma) \subset \mathrm{vars}(x\rho) \cup \mathcal{W}$. Note, that σ occurs here on both sides of the equation. During the recursive browsing of the context tree it may become necessary for the retrieval that the substitution σ binds index variables in $x\tau\rho$ as well as in $x\rho$. This is because of the fact, that a term in the context tree is represented by the composition of the substitutions along a path from the root to a leaf. Condition 3 in Definition 36 ensures that the algorithm has found an instance of the query once it has reached a leaf node. In the case of substitution trees, I refer to [Gra96] for the correctness proof. This proof can easily be extended to context trees. The implementation of the procedure InstTest is based on the procedure TermInst (Algorithm 7) that tests for two given terms s and t if s is an instance of t, i.e., if a substitution σ exist with $s\sigma = t\sigma$ and $\mathrm{dom}(\sigma) \in \mathrm{vars}(t) \cup \mathcal{W}$. The above note about InstTest also holds for TermInst, namely that σ can occur on both sides of the equation. In the case that s does not contain any variables from $\mathcal{W}$ this is the same as testing whether there is a substitution σ with $s = t\sigma$.

Algorithm 7: TermInst

Input: term s, term t, substitution σ

if $s \in \mathcal{W}$ **then return** $(true, \{s \mapsto t\})$;
if $t = x$ **then return** $(true, \{x \mapsto t\})$;
if $s = F(s_1, \ldots, s_n)$ *and* $t = f(t_1, \ldots, t_n)$ **then**
  foreach $i \in \{1, \ldots n\}$ **do**
    $(r, \sigma) = \mathrm{TermInst}(s_i, t_i, \sigma)$;
    if $r = false$ **then return** $(false, \emptyset)$;
  end
  if $F \in \mathrm{dom}(\sigma) \ \wedge \ F\sigma \neq f$ **then return** $(false, \emptyset)$;
  if $F\sigma = f$ **then return** $(true, \sigma)$ **else return** $(true, \sigma \circ \{F \mapsto f\})$;
end
return $(false, \emptyset)$;

Algorithm 8: InstTest

Input: substitution τ, substitution ρ

$\sigma = \emptyset$;
foreach $x \in \mathrm{dom}(\tau)$ **do**
  $(r, \sigma) = \mathrm{TermInst}(x\tau, x\rho, \sigma)$;
  if $r = false$ **then return** *(false,σ)*
end
return $(true, \sigma)$;

Maintenance Algorithms

The insertion and deletion operations are based on a modified version of Lookup. This lookup procedure, called LookupVariant (Algorithm 9), traverses a context tree for finding a variation of a term t encapsulated in the query substitution. In case there is no variation of t in the context tree, it, in addition to Lookup, returns a node that is a suitable position for inserting t into the context tree, if there is such a node in the context tree.

Variation The test procedure VariantTest (Algorithm 11) checks for a substitutions τ and a substitution ρ if for all $x \in \mathrm{dom}(\tau)$ $x\tau\rho\sigma = x\rho\sigma$ and $\mathrm{dom}(\sigma) \cap \mathcal{X} = \emptyset$. The procedure is depicted in Algorithm 11. This procedure uses a procedure (Algorithm 10) which tests for two given terms s and t if they are variations, i.e. $s\sigma = t\sigma$ and $\mathrm{dom}(\sigma) \cap \mathcal{X} = \emptyset$.

Algorithm 9: LookupVariant

Input: Context tree $T = (V, E, \mathrm{subst})$, $v \in V$, substitution ρ

```
1  HIT = ∅;
2  BEST = NULL;
3  foreach v' with (v, v') ∈ E do
4  |  if VariantTest(subst(v'), ρ) = (true, σ) then
5  |  |  if isLeaf(v') ∧ v_best = NULL then return (v', NULL, ρ ∘ σ);
6  |  |  (HIT, v_best, ρ') = LookupVariant(T, v', ρ ∘ σ, VariantTest);
7  |  |  if HIT then
8  |  |  |  return (HIT, NULL, ρ')
9  |  |
10 |  else if ∀x ∈ dom(subst(v')) top(x subst(v')) = top(xρ) and v_best = NULL then
11 |  |  v_best = v';
12 |  end
13 end
14 return (v, v_best , ρ);
```

The procedure LookupVariant (Algorithm 9) is invoked with a context tree T, a node v, and the query substitution ρ. Like Lookup (Algorithm 2), the node v is initially set to the root node of T, and it is the current examined node of T during the recursive application of LookupVariant. The substitution ρ is an accumulator argument, initially set to the substitution containing the term t to be inserted. It is the composition (line 6) of the initial query substitution and all substitutions σ computed in line 4 during the recursive application of LookupVariant. The procedure LookupVariant traverses the context tree T as long as the variant test (line 4) is successful. The algorithm of VariantTest is given in Algorithm 11. If the algorithm has found a leaf node (line 5) the recursion stops and it returns this leaf node. If VariantTest fails then LookupVariant checks if the terms in the codomain of the substitution of the current node and the

substitution ρ have the same top symbols (line 10). If they have the same top symbols then LookupVariant remembers this node in v_{best}. If no variant is found then the algorithm returns v_{best}. This node indicates a suitable position in the context tree T where a new leaf node can be created which represents t.

Algorithm 10: TermVariant

Input: term s, term t, substitution σ

```
1  if s = x ∧ s = t then ;
2  if s ∈ V_i then
3  |  if sσ = t then
4  |  |  return true
5  |  else if s ∉ dom(σ) then
6  |  |  σ = σ ∘ {s ↦ t};
7  |  |  return true;
8  |  else
9  |  |  return false;
10 |  end
11 end
12 if s = F(s_1, ..., s_n) and t = f(t_1, ..., t_n) then
13 |  foreach i ∈ {1, ... n} do
14 |  |  (r, σ) = TermVariant(s_i, t_i, σ);
15 |  |  if r = false then return (false, ∅);
16 |  end
17 |  if F ∈ dom(σ) ∧ Fσ ≠ f then return (false, ∅);
18 |  if Fσ = f then return (true, σ) else return (true, σ ∘ {F ↦ f});
19 end
20 return (false, ∅);
```

Algorithm 11: VariantTest

Input: substitution τ, substitution ρ

```
1  σ = ∅;
2  foreach x ∈ dom(τ) do
3  |  (r, σ) = TermVariant(xτ,xρ,σ);
4  |  if r = false then  return (false,σ)
5  end
6  return (true, σ);
```

Most Specific Common Generalization In order to insert a new term into the index it can become necessary to split a node. Therefore, a further operation is needed, namely the computation of the *most specific common generalization*; if τ and ρ are two substitutions and there exist substitutions σ_1 and σ_2 and μ such that $\mu \circ \sigma_1 = \tau$ and $\mu \circ \sigma_2 = \rho$, then μ is called a *common generalization*. If, additionally, there is a substitution δ for each other substitution $\nu \neq \mu$ such that $\mu = \nu \circ \delta$, then μ is called the *most specific common generalization* which is given by the following function:

$$\text{mscg}(\tau, \rho) := (\sigma_1, \sigma_2, \mu).$$

Insert The procedure EntryCreate inserts a term t into a context tree T. Remember, the variables of t are assumed to be normalized. First the term t is transformed into a query substitution ρ. Then EntryCreate calls LookupVariant with T, the root node v_r, and the query substitution ρ. Three cases can occur. The first is that LookupVariant has found a leaf (line 5) which represents t. Then a reference to t is inserted into the leaf node which is done by InsertReference. If there is no respective leaf node representing t then LookupVariant returns a node v_{best}, if there is one. The node v_{best} indicates a suitable insert position. In order to insert t into the index, EntryCreate first computes the $\text{mscg}(\text{subst}(v_{best}), \rho) = (\mu, \sigma_1, \sigma_2)$. After that, the procedure creates two new nodes v_1, v_2. All subnodes of v_{best} become subnodes of v_1 and are deleted from the subnodes of v_{best}. Then v_1 and v_2 become the new subnodes of v_{best} ($(v_{best}, v_1) \in E$ and $(v_{best}, v_2) \in E$). The substitutions of v_{best}, v_1 and v_2 are set to the substitutions computed by $\text{mscg}(\text{subst}(v_{best}), \rho)$ as follows: $\text{subst}(v_1) = \sigma_1$, $\text{subst}(v_2) = \sigma_2$ and $\text{subst}(v_{best}) = \mu$. After that, the path $v_r, \ldots, v_{best}, v_1$ represents the same terms as the former path $v_r, \ldots, v_{best}$. The path $v_r, \ldots, v_{best}, v_2$ represents the inserted term. Additionally, a reference to t is inserted into the leaf node v_2. The third case arises if none of the above occurs. This means, neither t has been inserted into the index before nor is there a suitable insert position. Then a new leaf node is inserted below v representing the term t.

Algorithm 12: EntryCreate

Input: Context tree $T = (V, E, \text{subst}, v_r)$, term t

$\rho = \{x_0 \mapsto t\}$;
if $\neg$ IsLeaf(v_r) **then**
  $(v, v_{best}, \rho') = \text{LookupVariant}(T, v_r, \rho)$;
end
if IsLeaf$(v) \wedge v_{best} = NULL$ **then** InsertReference(v, t) ;
else if $v_{best} \neq NULL$ **then**
  $(\sigma_1, \sigma_2, \mu) = \text{mscg}(\text{subst}(v_{best}), \rho')$;
  $V = V \cup \{v_1, v_2\}$;
  foreach $(v_{best}, v') \in E$ **do** $E = (E \setminus \{(v_{best}, v')\}) \cup \{(v_1, v')\}$;
  $E = E \cup \{(v_{best}, v_1), (v_{best}, v_2)\}$;
  InsertReference(v$_2$, t);
  $\text{subst}(v_{best}) = \mu$;
  $\text{subst}(v_1) = \sigma_1$;
  $\text{subst}(v_2) = \sigma_2$;
else
  $V = V \cup \{v'\}$;
  $E = E \cup \{(v, v')\}$;
  InsertReference(v', t);
end

Algorithm 13: EntryDelete

Input: Context tree $T = (V, E, \text{subst}, v_r)$, term t

```
1 ρ = {x_0 ↦ t};
2 if IsLeaf(v_r) then
3 |   RemoveReference(T, v, ρ)
4 else
5 |   (v', v_best) = LookupVariant(T, v', ρ);
6 |   if v' ≠ ∅ then RemoveReference(T, v', ρ);
7 end
```

Delete The procedure EntryDelete (Algorithm 13) removes the term t from the context tree T. It first generates the query substitution from t. If v_r is not a leaf node, EntryCreate applies LookupVariant in order to obtain the leaf node representing t. If there is such a leaf node in T then EntryDelete removes the reference to t from this node. EntryDelete does not delete a leaf node and collapse the context tree when a term is removed. It just removes its reference from the respective leaf. Although, it is possible to destruct the context tree after the deletion of a term, the indexing introduced in Chapter 5 does not do this. The reason for this is that the destruction of the context tree is too expensive in the context of reasoning about huge clause sets. In [Gra96] the destruction of the substitution tree indexing is shown which is analogous to the case of context trees. Not destructing a context tree requires a slight modification of the invariant of the original notions of context trees, i.e., each path in a context tree corresponds to a term stored in the index. The new invariant is as follows: a path represents a term stored in the context tree, if and only if the respective leaf node contains a reference to this term.

2.2.6. Rewrite Proofs for Transitive Relations

In this thesis, I consider ontologies represented in the BSH-Y2 first-order fragment (Chapter 4). The BSH-Y2 language contains transitivity axioms. The reasoning calculus for BSH-Y2 presented in Chapter 6, treats the transitivity axioms specially via the chaining calculus (Section 2.2.3).

Consequently, I use in this thesis the model construction of the chaining calculus presented in [BG98] which is minimal with respect to set inclusion. This model construction relies on the following depicted notions. The actual model construction is given in Chapter 4. For the proofs and further details, I refer to [BG98].

In this section, let N be a set of clauses without transitivity axioms, Tr be the set of transitive predicate symbols and $\mathcal{A}_{\mathrm{Tr}}$ be the transitive theory as already shown for the chaining calculus in Section 2.2.3 defined as follows.

$$\mathcal{A}_{\mathrm{Tr}} = \{Q(x,y), Q(y,z) \rightarrow Q(x,z) \mid Q \in \mathrm{Tr}\}$$

Definition 39 (Chain)**.** *A* chain *is a finite sequence of atoms*

$$Q(l_0,l_1), Q(l_1,l_2), \ldots, Q(l_{n-1},l_n)$$

where $n \geq 1$, all terms $l_0, \ldots, l_n$ are ground, and Q is a transitive predicate. The type *of such a chain is the atom $Q(l_0,l_n)$. A chain is called a* proof *in a Herbrand interpretation I if all atoms $Q(l_{i-1},l_i)$ are true in I. We say $Q(l_0,l_n)$ is provable in I if there exists a proof of type $Q(l_0,l_n)$ in I.*

Note, a subsequence of a proof is again a proof, and replacing a subproof with another subproof of the same type is again a proof.

Definition 40. *The* transitive closure *of a Herbrand interpretation I with respect to a transitive theory $\mathcal{A}_{\mathrm{Tr}}$ is defined as the set I together with all ground atoms $Q(l,r)$ which are logically implied in I by the transitivity axioms in $\mathcal{A}_{\mathrm{Tr}}$.*

Observation 41. *A Herbrand interpretation I is a model of a set of transitivity axioms $\mathcal{A}_{\mathrm{Tr}}$ if and only if it is identical to its transitive closure with respect to the transitive theory $\mathcal{A}_{\mathrm{Tr}}$.*

The following defines a rewrite system where each rewrite step defines one step in the proof for a transitive atom $Q(l,r)$, i.e., $Q \in \mathrm{Tr}$. Each step is defined with respect to the ordering of l and r

- $l \Rightarrow_Q r$ if $l \succ r$,
- $l \Leftarrow_Q r$ if $r \succ l$,
- $l \Leftrightarrow_Q r$ if $l = r$.

The annotation Q will be omitted if it is clear from the context or inessential.

Definition 42. *A* valley *is a chain of the form*

$$l_0 \Rightarrow l_1 \ldots \Rightarrow l_k \Leftarrow l_{k+1} \Leftarrow \ldots \Leftarrow l_n$$

or

$$l_0 \Rightarrow l_1 \ldots \Rightarrow l_k \Leftrightarrow l_{k+1} \Leftarrow \ldots \Leftarrow l_n$$

Valleys are also called rewrite proofs. *If an interpretation I contains a rewrite proof of type $Q(l,r)$ then this is denoted as $l \Downarrow_Q^I r$. A two step chain $l \Leftarrow t \Rightarrow r$ is called a* peak. *A chain $l \Leftrightarrow l \Rightarrow r$ or $l \Leftarrow r \Leftrightarrow r$ is called a* plateau. *A chain $l = l_0 \Leftrightarrow l_1 \Leftrightarrow \ldots \Leftrightarrow l_k = r$ is called a* plain *if $k \geq 2$.*

Definition 43. *A peak, plateau or plain* commutes *in the Herbrand interpretation I if there exists a rewrite proof of the same type in I. The* rewrite closure *of I with respect to a set of transitive predicates* Tr *is defined as $I \cup \{Q(l,r) : l \Downarrow_Q^I r \text{ and } Q \in \mathrm{Tr}\}$.*

Note, the rewrite closure is obviously contained in the transitive closure.

Definition 44 (Complexity of rewrite steps)**.** *We define*

- *the complexity of $l \Rightarrow_Q r$ as the multiset $\{l\}$,*
- *the complexity of $l \Leftarrow_Q r$ as the multiset $\{r\}$,*
- *the complexity of $l \Leftrightarrow_Q r$ as the multiset $\{l, r\}$.*

The complexity of a chain is the multiset of the complexities of all its individual steps.

Two chains are compared by their respective complexities in the two-fold multiset extension of the ordering $\succ$. The resulting ordering is denoted by $\succ_\pi$. Such an ordering on proofs can be called *proof ordering* as it satisfies the following properties:

- A proper subproof of a proof is smaller than the original proof.
- Replacement of any subproof by a smaller proof will result in a smaller proof.

Definition 45. *A proof of $Q(l,r)$ in I is said to be* minimal *(w.r.t. $\succ_\pi$) if there exists no smaller proof of the same type in I.*

Observation 46 (Characterization of minimal proofs)**.** *Let $\succ$ be a well-founded ordering on ground terms, $\succ_\pi$ be the corresponding proof ordering, and I be a Herbrand interpretation. If no peak, plateau, or plain in I is a minimal proof, then all minimal proofs in I are rewrite proofs. Furthermore, if a peak, plateau or plain commutes in I, then it is non-minimal.*

Lemma 47 (Commutation)**.** *Let $\succ$ be a well-founded and total ordering on ground terms. The rewrite closure of I with respect to a set of transitive predicates* Tr *is a model of $\mathcal{A}_{\mathrm{Tr}}$ if and only if all peaks in I commute.*

Definition 48. *The size of a rewrite proof*

$$Q(l_0, l_1), \ldots, Q(l_{n-1}, l_n)$$

with type $Q(l_0, l_n)$ in an interpretation I is defined as the length of the proof chain; written $|Q(l_0, l_n)|_I$.

3. The YAGO Ontology

The ontology YAGO (*Yet another great ontology*) [SKW07] is automatically generated out of Wikipedia and WordNet [Fel98]. It contains knowledge about the relations between individuals, classes and relations. Figure 3.1 shows a small excerpt from the knowledge of YAGO represented as a graph. This graph contains the information that Albert Einstein was a physicist and each physicist is a scientist. Further, Albert Einstein was born in Ulm and had three children: Eduard, Hans Albert, and Lieserl. The relation bornIn is a functional relation and the relation locatedIn is transitive.

This chapter gives a brief overview over the representation of the knowledge in YAGO. In order to apply first-order reasoning procedures, I have developed a translation of YAGO into a representation in first-order logic. The representation is designed in such a way that the reasoning procedures that I present in Chapter 6 and Chapter 7 perform efficiently with respect to this representation.

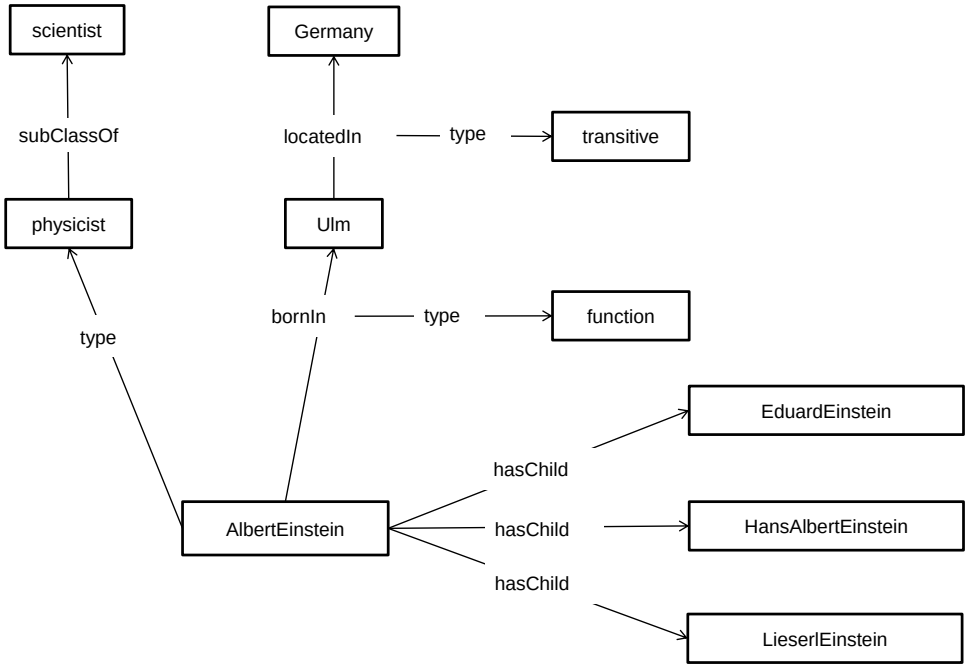

Figure 3.1.: YAGO Ontology

3.1. Knowledge Representation in YAGO

The YAGO ontology is automatically generated out of Wikipedia and WordNet [Fel98]. YAGO contains information about more than two million entities and has more than 15 million entries. A manual evaluation of YAGO by randomly choosing facts and comparing them with the respective Wikipedia page showed an accuracy of approximately 95% [SKW07]. The YAGO ontology is exceptional when compared to other ontologies because it is fully automatically generated, has a high coverage, and at the same time, a high accuracy rate.

The knowledge contained in YAGO is represented by facts. There are four types of facts in YAGO: facts about individuals, facts about classes, facts about relations, and facts about facts.

A fact in YAGO is a triple of the following form:

arg1 rel arg2,

where rel is a relation and arg1, arg2 are individuals, classes, facts, or relations. In addition, each fact has a unique id and a confidence value attached.

The triple representation of YAGO is a slight extension of the W3C standard representation format for ontologies RDF/XML [Bec04, Sta09].

The following gives examples for each type of fact. Facts about an individual are:

AlbertEinstein	bornIn	Ulm
AlbertEinstein	hasChild	EduardEinstein
AlbertEinstein	type	physicist.

These facts represent the knowledge about the individual 'Albert Einstein' saying that "Albert Einstein was born in Ulm", "Albert Einstein had a child called Eduard Einstein", and that "Albert Einstein was a physicist". A fact of the second type is:

physicist subClassOf scientist,

which states that every physicist is also a scientist. The third type of facts represents knowledge about relations of YAGO; for example,

bornIn	type	yagoFunction
locatedIn	type	yagoTransitiveRelation.

The first fact states that the relation bornIn is functional meaning that everybody has at most one birthplace. The second fact expresses that the relation locatedIn is transitive.

The last type of facts provides additional information to facts. This extra information expresses, for example, where a fact was extracted from and at what time this fact holds.

fact173859	foundIn	www. source. org
fact173859	until	1896

A detailed introduction to YAGO and its representation format can be found in [SKW07].

3.2. Translating YAGO into First-Order Logic

This section defines the clause set $N_{\mathcal{Y}}$ that represents the knowledge of the YAGO ontology in a first-order language over the signature $\Sigma_{\mathcal{Y}} = (\mathcal{F}_{\mathcal{Y}}, \mathcal{R}_{\mathcal{Y}})$. All occuring function symbols are constants, i.e., for all $c \in \mathcal{F}_{\mathcal{Y}}$ it holds that $\text{arity}(c) = 0$. During the translation, I distinguish four distinct clause sets: N_{fact}, N_{sort}, N_{trans}, and N_{func}. Each of them consists of a particular type of clauses: the set N_{fact} contains all the binary facts over individuals, N_{sort} contains the type and subtype information, N_{trans} all the transitivity axioms, and N_{func} the functionality constraints. For reasoning about YAGO, I do not consider the relations providing extra information like foundIn and until. I call the set of facts without this type of facts the *core* of YAGO. The representation of the core of YAGO in first-order logic is the union of the sets N_{fact}, N_{sort}, N_{trans}, and N_{func}. Additionally, I have implemented the presented translation, resulting in a tool that automatically transforms the core of YAGO into a representation in first-order logic.

Definition 49 (Fact)**.** *A positive ground unit clause $\rightarrow A$ is called a fact and a negative ground unit clause $A \rightarrow$ is called a negative fact.*

3.2.1. Relation of Individuals

Let arg1, arg2 be individuals and rel be a relation. For each fact about an individual

$$\text{arg1} \quad \text{rel} \quad \text{arg2},$$

there are constants c_{arg1}, $c_{\text{arg2}} \in \mathcal{F}_{\mathcal{Y}}$, a binary relation $P_{\text{rel}} \in \mathcal{P}_{\mathcal{Y}}$, and the unit clause

$$\rightarrow P_{\text{rel}}(c_{\text{arg1}}, c_{\text{arg2}}).$$

Table 3.2 shows all YAGO relations over individuals that are translated in this way. The resulting ground unit clauses are stored in N_{fact}.

3.2.2. Classifying Individuals

Facts that have an individual as their first argument and a class as their second argument are of the following form:

actedIn	bornIn	created
dealsWith	diedIn	directed
discovered	graduatedFrom	happenedIn
hasAcademicAdvisor	hasCapital	hasChild
hasCurrency	hasOfficialLanguage	hasPredecessor
hasProduct	hasProductionLanguage	hasSuccessor
hasWonPrize	inTimeZone	influences
interestedIn	isAffiliatedTo	isCitizenOf
isLeaderOf	isMarriedTo	livesIn
locatedIn	madeCoverFor	originatesFrom
participatedIn	politicianOf	produced
worksAt	wrote	

Figure 3.2.: Relations of individuals

arg1	type	arg2
arg1	isOfGenre	arg2
arg1	musicalRole	arg2.

For all of these facts, there are constant $c_{\text{arg1}} \in \mathcal{F}_{\mathcal{Y}}$, the monadic (sort) predicate $S_{\text{arg2}} \in \mathcal{P}_{\mathcal{Y}}$, and the clause

$$\rightarrow S_{\text{arg2}}(c_{\text{arg1}}),$$

which is added to N_{sort}.

3.2.3. Relations of Classes

Subclass relations are expressed by the following types of facts in YAGO:

arg1	subClassOf	arg2
arg1	isMemberOf	arg2
arg1	isSubstanceOf	arg2.

Each fact of this type is translated into a subsort relation and added to the set N_{sort}.

$$S_{\text{arg1}}(x) \rightarrow S_{\text{arg2}}(x)$$

where S_{arg1} and S_{arg2} are sorts with $S_{\text{arg1}}, S_{\text{arg2}} \in P_{\mathcal{Y}}$, and $x \in \mathcal{X}$ is a variable.

Translating classes into sorts instead of translating them to constants has the advantage that reasoning about sorts can be efficiently performed independently from the

remaining reasoning tasks via special data structures and algorithms. More details about sort reasoning in the context of large ontologies are presented in Chapter 6 and in Chapter 7. In [Wei01, SS89] you find further details about reasoning about sorts in general. Note, the sort theory that we get by this translation of YAGO is acyclic and static.

3.2.4. Functionality Constraint

There are facts in YAGO stating that a relation is functional. For example, the following fact states that the relation rel is a functional relation:

rel type yagoFunction.

These kind of facts seem to be second order but they can be translated into their respective first-order axiom: the functionality axiom. The relation rel is represented as a predicate $P_{\text{rel}} \in \mathcal{P}_{\mathcal{Y}}$ and the following functionality axiom is added to N_{func}:

$$\mathcal{P}_{\text{rel}}(x, y), \mathcal{P}_{\text{rel}}(x, z) \rightarrow y \approx z,$$

where $x, y, z \in \mathcal{X}$ are variables.

3.2.5. Transitivity Axiom

In YAGO, there are facts expressing that a relation rel is a transitive relation:

rel type yagoTransitiveRelation.

Like in the case of functions, these facts are translated into the respective first-order transitivity axioms:

$$P_{\text{rel}}(x, y), P_{\text{rel}}(y, z) \rightarrow P_{\text{rel}}(x, z),$$

where $x, y, z \in \mathcal{X}$ are variables. The axiom is added to the set N_{trans}.

3.2.6. Unique Name Assumption

In addition, there is an implicit *unique name assumption* for YAGO. Expressed in first-order logic this means that for two syntactically different constants $c_1, c_2 \in \mathcal{F}$ and for all Herbrand models I the following holds: $I \models \neg(c_1 \approx c_2)$. The set N_{una}, which states the unique name assumption, is defined as the following set of clauses:

$$\text{N}_{\text{una}} = \{a \approx b \rightarrow \ \| \ a, b \in \mathcal{F}_{\mathcal{Y}} \text{ and } a \neq b\}$$

3.2.7. First-Order Representation of YAGO

The previous sections showed how the different types of facts are translated into a first-order representation. The YAGO ontology represented in first-order language is the set $N_{\mathcal{Y}}$ that is the union of all the previously defined sets

$$\mathrm{N}_{\mathcal{Y}} = N_{\mathrm{fact}} \cup N_{\mathrm{sort}} \cup N_{\mathrm{func}} \cup N_{\mathrm{trans}} \cup \mathrm{N}_{\mathrm{una}}$$

The clause set $\mathrm{N}_{\mathcal{Y}}$ consists of clauses from the Bernays–Schönfinkel Horn fragment with equality. Note, $\mathrm{N}_{\mathcal{Y}}$ does not represent the whole knowledge of YAGO. The translated part consists of the core knowledge that I consider in this thesis. The signature $\Sigma_{\mathcal{Y}}$ consists of about 2 million elements and $\mathrm{N}_{\mathcal{Y}}$ consists of about 10 million clauses.

Facts that give additional information to facts like the relation foundIn, are not considered for reasoning in this work and therefore are not translated into first-order logic.

In addition, I have omitted facts of YAGO that could not be translated into Horn clauses or that contain time and date information. Integrating this type of facts into the reasoning procedure presented in this thesis, is left for future work.

4. Reasoning in BSH-Y2

In this chapter, I define the ontology language BSH-Y2 which is a decidable fragment of first-order logic. Furthermore, I present the first-order reasoning problems that correspond with checking the consistency of an ontology and answering queries in an ontology.

The language BSH-Y2 can represent the knowledge of large ontologies. Additionally, the reasoning procedures developed in Chapter 6 and Chapter 7 perform efficient on large sets of this language. In particular, the language BSH-Y2 contains the first-order representation of YAGO as presented in the last chapter and it represents large parts of the two ontologies: SUMO [NP01a] and CYC [Len95].

4.1. Defining BSH-Y2

The language BSH-Y2 is an extension of the language representing the YAGO ontology. It contains constraints and defined relations in addition to the clauses presented in Chapter 3. Furthermore, BSH-Y2 is a subset of the Bernays–Schönfinkel Horn fragment and, consequently, decidable.

The following defines properties of Horn clauses and sets of Horn clauses N.

Definition 50 (Range restricted)**.** *A Horn clause* $\Gamma \to A$ *is range restricted if and only if* $\mathrm{vars}(A) \subseteq \mathrm{vars}(\Gamma)$.

Definition 51 (Defined predicate)**.** *A range restricted Horn clause* $C = \Gamma \to P(t_1, \ldots, t_n)$ *with* $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ *and* $P \in \mathcal{R}$ *is called a definition for predicate* P*. In other words,* P *is defined by* C*.*

Definition 52 (Dependent definitions)**.** *The dependency of a definition* $C = \Gamma \to A$ *on a set* $D \subset \mathcal{R}$ *in a Horn clause set* N *is inductively defined. The definition* C *is dependent on* D *iff there is a predicate symbol* P *in* Γ *with (i)* $P \in D$ *or (ii)* P *is defined by a clause* $C' \in N$ *that is dependent on* D*. Otherwise, the clause* C *is independent from* D *in* N*.*

Definition 53 (Transitive dependent)**.** *A clause* C *is transitive dependent in a clause set* N *iff* C *is dependent on* Tr*. The set* Tr *is the set of transitive predicate symbols occurring in* N*.*

Definition 54 (Acyclic definitions)**.** *A definition* $C = \Gamma \to P(t_1, \ldots, t_n)$ *is acyclic if it is independent from every set* D *with* $P \in D$*.*

4.1.1. Constraints

Constraints are clauses of the following form

$$P_1(t_{11}, \ldots, t_{1n_1}), \ldots, P_k(t_{k1}, \ldots, t_{kn_k}) \rightarrow$$

This allows to formulate negative facts like *Albert Einstein was not born in Munich*:

$$\text{bornIn}(\text{AlbertEinstein}, \text{Munich}) \rightarrow$$

Additionally, constraints like asymmetry and irreflexivity can be expressed for a relation rel:

$$P_{\text{rel}}(x, x) \rightarrow$$
$$P_{\text{rel}}(x, y), P_{\text{rel}}(y, x) \rightarrow$$

4.1.2. Defined Relations

Defined relations provide a method to define a predicate in relation to other predicates. A defined relation in a clause set N is a defined predicate as follows:

$$C = P_1(t_{11}, \ldots, t_{1n_1}), \ldots, P_k(t_{k1}, \ldots, t_{kn_k}) \rightarrow P(s_1, \ldots, s_m)$$

with (i) C is acyclic in N, and (ii) if P is transitive then C is transitive independent in N.

The requirement that defined relations are acyclic is needed because the query answering procedure does not remove redundant clauses. The requirement (ii) ensures that the saturation procedure terminates.

The definition of relations is often used in the specification of ontologies. For example, the relation hasSon can be specified in terms of the relations male and hasChild.

$$\text{male}(y), \text{hasChild}(x, y) \rightarrow \text{sonOf}(y, x)$$

Transitive Relations and Constraints

Although the relation locatedIn is transitive, defined relations allow us to formulate constraints like irreflexivity and asymmetry for the relation locatedIn without a complete ground instantiation of locatedIn. This can be achieved by the definition of a new predicate locatedInTC representing the transitive closure of locatedIn. The resulting clauses are

$$\text{locatedIn}(x, x) \rightarrow \tag{4.1}$$
$$\text{locatedIn}(x, y), \text{locatedIn}(y, x) \rightarrow \tag{4.2}$$

$$\text{locatedIn}(x, y) \rightarrow \text{locatedInTC}(x, y) \tag{4.3}$$
$$\text{locatedInTC}(x, y), \text{locatedInTC}(y, z) \rightarrow \text{locatedInTC}(x, z) \tag{4.4}$$

Equation 4.1 and Equation 4.2 express the irreflexivity and asymmetry constraints, the Equation 4.3 is the definition of the new predicate locatedInTC, finally, Equation 4.4 is the transitivity axiom for locatedInTC.

4.1.3. BSH-Y2

The ontology language BSH-Y2 is a subset of the Bernays–Schönfinkel Horn fragment with equality. This language can express the core of the YAGO ontology and supports additional constructs like constraints and defined relations. Chapter 8 shows an extension of YAGO by constraints and defined relations called YAGO++. Further, it shows that large parts of the ontologies SUMO and CYC can also be expressed in BSH-Y2. In addition, the reasoning procedures I have developed in this thesis work efficient for huge clause sets from BSH-Y2. Figure 4.1 depicts the type of clauses of BSH-Y2.

$\rightarrow P(a_1, \ldots, a_n)$	Ground Fact
$S(x) \rightarrow T(x)$	Subsort Relation
$R(x, y), R(x, z) \rightarrow y \approx z$	Functionality Axioms
$R(x, y), R(y, z) \rightarrow R(x, z)$	Transitivity Axioms
$P_1(t_{11}, \ldots, t_{1n_1}), \ldots, P_k(t_{k1}, \ldots, t_{kn_k}) \rightarrow$	Constraints
$P_1(t_{11}, \ldots, t_{1n_1}), \ldots, P_k(t_{k1}, \ldots, t_{kn_k}) \rightarrow P(s_1, \ldots, s_m)$	Defined Relations

Figure 4.1.: BSH-Y2

where a_i are constants, $x, y, z \in \mathcal{X}$ are variables, each t_{ij}, s_i is either a constant or a variable. The symbol R denotes a binary predicate, S_i denotes a sort predicate, and P, P_i denote predicates with arbitrary arity. The predicates of the functionality axioms are non-transitive and not defined. Note, defined relations are acyclic and transitive independent if P is transitive in a BSH-Y2 clause set.

Note, for the static sort theory S_N of clause sets N from BSH-Y2 it holds that $\mathrm{S}_N \subseteq N$. This means the sort theory of N is exactly its static sort theory. The reason for this is that a positive sort atom occurs only in monadic facts ($\rightarrow S(a)$) or in subsort relations ($S_1(x) \rightarrow S_2(x)$) in N. In addition, assume that the sort theory S_N contains only acyclic subsort relations. If S_N contains a cycle then this cycle can be deleted from S_N.

4.2. Reasoning in BSH-Y2 ontologies

In this thesis, I focus on two reasoning tasks for ontologies represented as clause sets from BSH-Y2. The first task is verifying whether a particular ontology is consistent, i.e., whether it is satisfiable. Formally, if N is a clause set from BSH-Y2 then satisfiability checking is the following reasoning problem

$$N \not\models \bot \tag{4.5}$$

The second reasoning task is query answering in the minimal model. Given a first-order formula Φ and a minimal model N_I of a BSH-Y2 ontology N. Then query answering in the minimal model means verifying whether the formula Φ is entailed by the minimal model N_I. Formally, this is the following reasoning problem.

$$N_I \models \Phi \tag{4.6}$$

In the context of huge ontologies like YAGO, standard first order reasoning procedures are not suited to perform these reasoning tasks. This has two reasons. First, the standard first-order reasoning framework is too prolific. Second, minimal model reasoning is beyond standard first-order reasoning; in particular, if queries with quantifier alternations are considered. The following provides further details about these reasoning problems.

4.2.1. Satisfiability

Before being able to answer queries in the minimal model, the clause set representing the ontology has to be verified to be satisfiable, i.e., $N \not\models \bot$. This is because, the minimal model exists only if the clause set is satisfiable. Verifying the satisfiability of a BSH-Y2 ontology can, in principle, be accomplished by a finite saturation.

However, the BSH-Y2 is a subset of the the Bernays–Schönfinkel Horn fragment with equality. Checking satisfiability of a clause set from this fragment is EXPTIME complete [Pla84]. Therefore, standard general purpose reasoning procedures are not feasible for practically verifying the satisfiability of huge ontologies which consists of several million clauses. The experiments presented in Chapter 8 verify this.

Furthermore, already the existence of one defined relation (Section 4.1.2) containing a transitive atom causes the standard reasoning procedure to blow up the search space. This phenomenon is illustrated in Chapter 6.

In Chapter 6, I present a reasoning procedure that is based on the superposition reasoning framework and verifies the consistency of BSH-Y2 ontologies by a finite saturation in less than one hour (Chapter 8).

4.2.2. Query Answering in the Minimal Model

Minimal Model Semantics

The minimal model semantics corresponds to a closed world assumption which contrasts the open world assumption of standard first-order semantics. In minimal model semantics, every formula Φ is assumed to be true if it is entailed by the minimal model N_I of a clause set N and is assumed to be false, otherwise. This is in contrast to standard semantics where the formula Φ is assumed to be true if it is entailed by all models I of N, i.e. $I \models N$ implies $I \models \Phi$.

The next example illustrates the differences between these two semantics.

Example 55 (Minimal model semantics)**.** *For example, consider the following clause set:*

$$N' = \{\rightarrow P(a), P(a) \rightarrow Q(a)\},$$

the clause $C = P(x) \rightarrow Q(x)$*, and the model* $I' = \{P(a), Q(a), P(b)\}$*. In this case,* I' *is a model of* N'*, i.e.,* $I' \models N'$ *but it is not a model of* C*, i.e.* $N' \not\models C$*.*

However, I' *is not minimal with respect to set inclusion because* $N_I = \{P(a), Q(a)\}$ *is also a model of* N'*,* $N_I \models N'$ *and also* $N_I \models C$*. Consequently, the clause* C *holds with respect to minimal model semantics but not with standard semantics.*

The BSH-Y2 language contains only Horn clauses and, therefore, there is a unique minimal model for every BSH-Y2 ontology. Also in [Rei77a, Rei77b] it is shown that one can make the closed world assumption for a set of Horn clauses. An extension of the closed world assumption for non-Horn clauses is developed in [Min82]. The closed world assumption is also called *negation as failure* [Cla78].

The following defines a minimal candidate interpretation for a set N consisting of clauses from BSH-Y2. The construction of a minimal candidate interpretation for a clause set from BSH-Y2 follows the construction of [BG98]. The chaining calculus which I consider in this thesis is sound and complete in terms of this model construction. I have extended this by a selection function as suggested in [BG01]. The saturation calculus that I present in Chapter 6 is sound and complete with respect to the following model construction. The respective proofs are provided in Chapter 6, too.

Definition 56 (Candidate interpretation)**.** *Let* N *be a set of clauses from the BSH-Y2 without transitivity axioms such that the transitive predicates of* N *are in the set* Tr*. Further, let* $\succ$ *be an admissible ordering. The following defines a candidate interpretation for* N *and* Tr*. Let* $C = \Gamma \rightarrow A$ *be a ground instance of a clause from* N*. Suppose* $E_{C'}$ *and* $R_{C'}$ *have been defined for all ground clause* C' *with* $C \succ C'$*. Then*

$$R_C = \bigcup_{C \succ C'} E_{C'}$$

if (i) $A \succ \Gamma$*, (ii)* $A \notin R^*_C$*, (iii)* $\Gamma \subseteq R^*_C$*, and (iv) no literal is selected in* C *then*

$$E_C = \{A\}$$

otherwise $E_C = \emptyset$*. If* $E_C \neq \emptyset$*, we say that* C *is productive and produces* A*.*

$$R^*_C = R_C \cup \{Q(l, r) : l \Downarrow^{R_C}_Q r \wedge Q \in \mathrm{Tr}\}$$

The interpretation N_I *of* N *is defined as* $N_I = \bigcup_C R^*_C$*.*

Note, this definition is also defined for sets containing non-ground Horn clauses via the lifting lemma which is a standard result of the superposition framework.

Lemma 57 (Monotonicity)**.** *For each clause* C *if* $R^*_D \models C$ *for a clause* $D \succ C$ *then for all* $D' \succ D$ *also* $R^*_{D'} \models C$*.*

Query Language

In order to answer sophisticated queries efficiently, I have identified a query language as a subset of the first-order language. The query language, which I introduce in Chapter 7, contains queries with arbitrary quantifier alternations. The respective query answering procedure efficiently answers the queries of this language. For example, the following query is contained in this query language:

$$\exists x, y(\mathrm{physicist}(x) \wedge \mathrm{bornIn}(x, y) \wedge \forall z(\mathrm{hasChild}(x, z) \rightarrow \mathrm{bornIn}(z, y))) \tag{4.7}$$

Further, all queries of Figure 1.2 are also elements of the query language. In addition to its expressiveness, the query language should have the property that its queries can be efficiently answered using appropriate reasoning procedures. The query answering procedure I present in Chapter 7 answers queries formulated in this language with respect to minimal model semantics.

Query answering in the Minimal Model

As defined above, answering a query Φ in the minimal model N_I of a BSH-Y2 ontology N is the following reasoning task:

$$N_I \models \Phi. \tag{4.8}$$

In general, reasoning in minimal models is beyond standard first-order reasoning. In particular, some of the queries of Figure 1.2 cannot be answered with these reasoning procedures. If considering simple existential queries then the standard first-order reasoning is equivalent to minimal model reasoning.

Proposition 58. *[HW08] For a Horn clause set N and queries of the form $\exists\vec{x}.\Gamma$ where Γ is a conjunction of literals it holds that*

$$N_I \models \exists\vec{x}.\Gamma \Leftrightarrow N \models \exists\vec{x}.\Gamma$$

A set of clauses from the BSH-Y2 contains only Horn clauses and, consequently, has a unique minimal model. Furthermore, the minimal model is finite because BSH-Y2 is a subset of the Bernays–Schönfinkel fragment. Consequently query answering in terms of the minimal model is decidable. The problem is that the query cannot be transformed into a clausal representation by Skolemization because this changes the domain. In [HW08] it is shown that already fixing a domain leads to reasoning tasks that go beyond first-order reasoning.

As a result, all queries not having the above form need a special query answering procedure. In Chapter 7, I introduce a new efficient, sound and complete, query answering procedure that answers complex queries in BSH-Y2 ontologies with respect to minimal model semantics. Instead of applying a Skolemization, the procedure performs a specific finite quantifier elimination algorithm. This algorithm works with respect to the saturation of the ontology. Chapter 7 proves that the saturation of a BSH-Y2 ontology with the calculus of Chapter 6 is a sufficient and efficient representation of the minimal model for this purpose. Chapter 8 evaluates the performance of the query answering procedure and shows its efficiency for practical query answering.

4.3. Summary

This chapter has presented BSH-Y2 which is a subset of the Bernays–Schönfinkel first-order fragment with equality. The BSH-Y2 fragment can encode the knowledge contained in YAGO and it provides additional types of clauses, namely constraints and defined relations. So it provides additional constructs to extend the knowledge of the YAGO ontology. Such an extension, called YAGO++, is defined in Chapter 8. The BSH-Y2 language can encode large parts of further ontologies like SUMO and CYC. In the remainder of this thesis, I present the first superposition based procedures which are able to saturate ontologies consisting of several million clauses from BSH-Y2 and answer queries with respect to minimal model semantics.

Reasoning in BSH-Y2 is complicated especially if considering ontologies consisting of several million clauses. The experiments presented in Chapter 8 show that standard reasoning procedures are too prolific. In addition, answering complex queries containing arbitrary quantifier alternations, in terms of the minimal model leads to reasoning tasks that are beyond standard first-order reasoning.

5. Filtered Context Tree Term Indexing

The invention of term indexing data structures has been pivotal for the success of automated theorem proving because they provide effective retrieval operations for formulas. Therefore, they build the basis for efficient implementations of the inference and reduction rules of the superposition reasoning framework. The inference and reduction rules access the term index several thousand times during the application of one single reasoning loop. For successful reasoning about ontologies that consist of several million clauses, it was necessary to further advance the term indexing techniques. I have developed a sophisticated term index and the respective algorithms which I present in this chapter. The new term index, called *filtered context tree index* [SWW10], has been the key for successful reasoning about ontologies like YAGO that consists of about 10 million clauses.

Filtered context tree indexing extends context tree indexing [GNN01] (Section 2.2.5) with a filtering mechanism. For example, terms of the following form occur in the YAGO ontology: $Q(a, b)$, $Q(a, x)$, $Q(x, b)$, $Q(x, y)$, $S(a)$, and $S(x)$, where Q is a binary predicate symbol, a, b are constants and S is a monadic predicate (sort symbol) from the signature. Performing retrieval operations fast on an index containing such atoms, requires the procedure to efficiently filter out subtrees of the index that do not lead to a success with respect to the current retrieval operation.

The filtered context tree indexing is the key for the efficient implementation of the saturation procedure in Chapter 6 and the query answering procedure in Chapter 7. The automated theorem prover SPASS [WDF+09] implements substitution tree indexing which is an instance of context tree indexing. For evaluating the filtering techniques presented in this section, I have integrated these into the substitution tree index of SPASS. The version of SPASS containing the new index and the implementation of the procedures presented in Chapter 6 and Chapter 7 is called SPASS–Y2. The evaluation in Chapter 8 confirms that SPASS–Y2 efficiently saturates BSH-Y2 ontologies, like YAGO and answers queries in terms of minimal model semantics. SPASS without the implementation of the new indexing techniques was even unable to load these clauses into the index within reasonable time. Additionally, this chapter presents implementation details of the index and further improvements.

When performing a retrieval operation, the procedure Lookup (Algorithm 2) pursues paths that do not contribute to the current query. In the case of ontologies like YAGO, this approach is not feasible because one subnode may have millions of subnodes and the term indexing is processed several thousand times in a single reasoning loop. Therefore, I have developed a mechanism that efficiently filters out subtrees of a context tree index whose paths do not contribute to the current query.

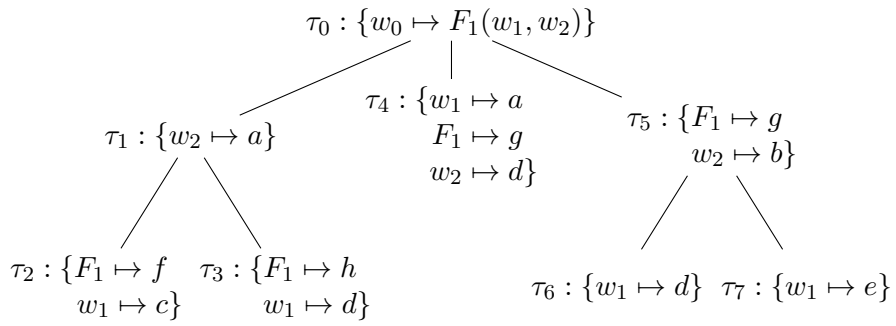

Figure 5.1.: Context Tree

The following example demonstrates a retrieval operation on the context tree depicted in Figure 5.1.

Example 59. *Consider the context tree of Figure 5.1 and the retrieval of terms unifiable with the term $g(e, x)$. The query substitution ρ for $g(e, x)$ is $\rho = \{w_0 \mapsto g(e, x)\}$. The algorithm starts with the query substitution ρ at the root node whose substitution is τ_0. The substitution τ_0 is unifiable with ρ using the following substitution:*

$$\sigma = \{w_1 \mapsto e, w_2 \mapsto x, F_1 \mapsto g\}.$$

Further descending the index requires to check all subnodes. In this case, these are the nodes containing τ_1, τ_4 and τ_5. Unifiable under the current substitution $\rho \circ \sigma$ are the substitutions τ_1 and τ_5. At first, the algorithm proceeds by inspecting the subtree starting at the node with τ_1. The substitution τ_1 is unifiable with $\rho \circ \sigma$ using $\sigma' = \{x \mapsto a\}$. Continuing with the subnodes, the algorithm recognizes that neither τ_2 nor τ_3 are unifiable with $\rho \circ \sigma \circ \sigma'$. Then the algorithm backtracks, proceeds with τ_5 and eventually finds a leaf where all substitutions along the path τ_0, τ_5, τ_7 are unifiable under the respective substitution ρ and returns the desired term which is $w_0\tau_0\tau_5\tau_7$.

In this example, after examining the node containing the substitution τ_0, the retrieval procedure proceeds by examining all subnodes. These subnodes are the nodes containing the substitutions τ_1, τ_4 and τ_5. Looking at the query, the symbol g has to occur in a substitution of some node along a successful path. However, if we inspect the subtree starting at the node with the substitution τ_1, we recognize that the symbol g does not occur in any substitution of this subtree. Consequently, this subtree does not have a successful path and can, therefore, be excluded from further processing.

The following presents the filtering technique in detail and shows the respective retrieval operations. Section 5.1, introduces filtered context trees and Section 5.2 presents the algorithms for the retrieval operations of filtered context trees and its soundness and completeness proofs. Some details about the implementation of these techniques in SPASS-Y2 can be found in Section 5.3 and further possible optimization in Section 5.4.

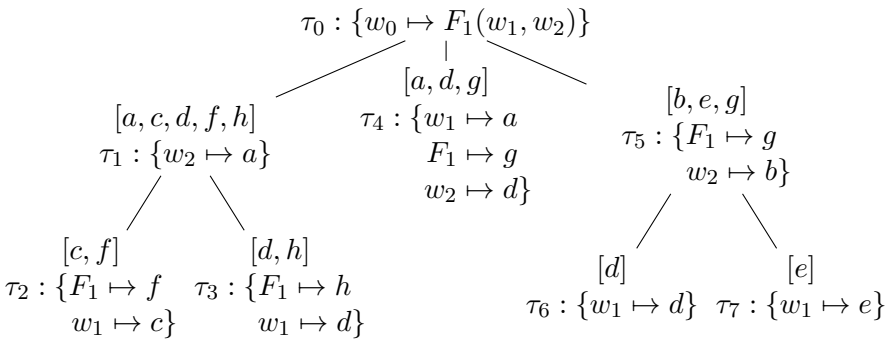

Figure 5.2.: Filtered Context Tree

5.1. Filtered Context Trees

In order to obtain filtered context trees, the notion of context trees has to be extended. In addition to context trees, filtered context trees contain a mapping M that maps each node v and any symbol s to each subnode of v that contains s in one substitution along a path starting at v including v itself. In order to be more efficient, M considers only the symbols that are characteristic for a substitution instead of considering all symbols that occur in a substitution. The following introduces the notion of a characteristic function for a substitution. More precisely, it defines what the characteristic symbols of a substitution are.

Example 60. *Reconsider Example 59 with the unification retrieval operation for the query substitution $\rho = \{w_0 \mapsto g(e, x)\}$. Extending Figure 5.1 with the map M yields the filtered context tree depicted in Figure 5.2. The retrieval algorithm applied to Figure 5.2 examines the node containing the substitution τ_0. As we have seen only those subtrees can contribute to the current retrieval operation that contain g in the codomain of the substitution of any of its nodes. The function M contains exactly this information. If we apply g to M, the function M returns those subtrees; in this example, these are the subtrees starting at the nodes containing the substitution τ_4 and τ_5. Consequently, the node containing the substitutions τ_1 is not considered during the retrieval.*

A mapping mechanism has also been used for discrimination trees. In discrimination tree indexing, the mapping assigns to a given label the respective successor node of the discrimination tree. For example, this has been added to the index of the theorem prover E [Sch02].

Using a mapping mechanism for context trees requires a function that assigns a set of signature symbols to a given substitution. The following defines such a function which is called the *characteristic function*. The characteristic function assigns to a substitution the set of top symbols occurring in its codomain. In order to also characterize a substitution containing only variables in its codomain, assume a symbol $\perp$ with $\perp \notin \Sigma$.

Definition 61 (Characteristic function)**.** *Let σ be a substitution and $\mathcal{O} \subset \mathcal{X} \cup \mathcal{U} \cup \mathcal{W}$ be a finite set of variables. The set of top symbols of σ respecting $\mathcal{O}$ is defined as*

$$ts(\sigma, \mathcal{O}) = \{f \mid \exists x \in \mathrm{dom}(\sigma) \cap \mathcal{O} \text{ with } x\sigma = f(t_1, \ldots, t_n)\}$$

The characteristic function $\mathrm{chr}(\sigma, \mathcal{O})$ *of a substitution σ with respect to the set of variables $\mathcal{O}$ is defined as follows:*

$$\mathrm{chr}(\sigma, \mathcal{O}) = \begin{cases} ts(\sigma, \mathcal{O}) & \text{if } ts(\sigma, \mathcal{O}) \neq \emptyset \\ \{\bot\} & \text{if } ts(\sigma, \mathcal{O}) = \emptyset \wedge \exists x \in \mathrm{dom}(\sigma) \text{ with} \\ & \quad x\sigma \in \mathcal{X} \vee x\sigma \in \mathcal{T}(\Sigma \cup \mathcal{U}, \mathcal{X}) \setminus \mathcal{T}(\Sigma, \mathcal{X}) \vee x \in \mathcal{X} \\ \emptyset & \text{otherwise} \end{cases}$$

Note, this definition also includes the cases where $x\sigma$ is a constant or $x\sigma$ is a function symbol mapped from a function variable.

Example 62. *Reconsider the query substitution $\rho = \{w_0 \mapsto g(e, x)\}$ of Example 59. The characteristic function of ρ is* $\mathrm{chr}(\rho, \{w_0\}) = \{g\}$*. Note that g is the only symbol of the characteristic function of ρ because this is the top symbol of the term $g(e, x)$. A term that is unifiable with $g(e, x)$ is of the form $g(y, x)$, where y is either a variable or the constant e. Consequently, the symbol g is the only symbol characterizing ρ.*

Once the characteristic function for a substitution has been defined, a context tree can be extended with a function M that assigns to a given node v and a symbol s a set of successor nodes. For each node v' in the set of successor nodes it holds that there is a node on a path, starting at v', that contains the symbol s in the characteristic of its substitution. On the other hand, the characteristic of substitutions that do not match the characteristic of a subtree in the context tree can be excluded from the current retrieval process. Matching the characteristic functions is a necessary requirement for a successful retrieval operation. This lifts the characteristic function of a substitution of one node to the characteristic of a subtree of a context tree.

Definition 63 (Filtered Context Tree)**.** *A* filtered context tree
$FT = (V, E, \mathrm{subst}, v_r, M, \mathrm{succ})$ is a context tree $(V, E, \mathrm{subst}, v_r)$ together with a function $M : V \times (\Sigma \cup \{\bot\}) \to 2^V$ from nodes and function symbols to a subset of V such that $v_{k+1} \in M(v_k, s)$ iff there is a path $v_1, \ldots, v_k, v_{k+1}, \ldots, v_n$, where v_1 is the root node v_r, with

$$s \in \bigcup_{i \in \{k+1, \ldots, n\}} \mathrm{chr}(\mathrm{subst}(v_i), \mathrm{vars}(v_1, \ldots, v_k)).$$

The function $\mathrm{succ}\,(v, \rho, \mathcal{O})$ *defines the set of successor nodes of $v \in V$ with respect to a substitution ρ and a set of variables $\mathcal{O} \subset \mathcal{X} \cup \mathcal{U} \cup \mathcal{W}$ as follows:*

$$\mathrm{succ}\,(v, \rho, \mathcal{O}) = \begin{cases} \{v' \mid (v, v') \in E\} & \text{if } \mathrm{chr}(\rho, \mathcal{O}) = \{\bot\} \\ \bigcup_{s \in \mathrm{chr}(\rho, \mathcal{O}) \cup \{\bot\}} M(v, s) & \text{otherwise} \end{cases}$$

Algorithm 14: FilteredLookup

Input: $FT = (V, E, \text{subst}, v_r, M, \text{succ})$, $v \in V$, substitution ρ, function Test

```
1 HITS = ∅;
2 if Test(subst(v), ρ) = (true, σ) then
3 |   if isLeaf(v) then return {v};
  |   /* v_r = v_1, ..., v_n = v path from the root v_r to v_n */
4 |   foreach v' ∈ succ(v, ρ, vars(v_1, ..., v_n)) do
5 |   |   HITS = HITS ∪ FilteredLookup(FT, v', ρ ∘ σ, Test);
6 |   end
7 end
8 return HITS
```

5.2. Algorithms for Filtered Context Trees

The procedure FilteredLookup (Algorithm 14) depicts the function performing the lookup operation on a given filtered context tree FT, a starting node v, a query substitution ρ and a function Test. The node v is the current examined node of FT during the recursive application of FilteredLookup. Initially, the node v is the root node v_r. The function Test is either UnifyTest (Algorithm 4), GenTest (Algorithm 6), or InstTest (Algorithm 8). These are the standard algorithms for the test functions shown in Section 2.2.5. These test functions expect two substitutions as their argument and are, therefore, independent from the underlying index. As a result, the standard algorithms can also be used for filtered context trees.

FilteredLookup (Algorithm 14) is almost the same algorithm as Lookup (Algorithm 2). The only difference is line 4. Instead of considering all children of v, FilteredLookup considers only the nodes which are returned by succ. The set

$$\text{succ}(v, \rho, \text{vars}(v_1, \ldots, v_n))$$

contains only those nodes which match the characteristic of ρ. Computing the characteristic of ρ is in time $O(|\operatorname{dom}(\rho)|)$, where $|\operatorname{dom}(\rho)|$ is the number of elements of the domain of ρ. As a result, the time complexity of the function succ is in time $O(|\operatorname{dom}(\rho)| * \log |\Sigma|)$ where $|\Sigma|$ is the number of symbols in the signature.

The algorithms for insertion, EntryCreate (Algorithm 12), and deletion, EntryDelete (Algorithm 13), use the procedure LookupVariant (Algorithm 9). The procedure LookupVariant has to be modified analogously to FilteredLookup (Algorithm 14) due to the fact that LookupVariant is a variation of Lookup (Algorithm 2).

Additionally, the procedure EntryCreate (Algorithm 12) has to maintain the map M when inserting a term into the indexing. When the procedure inserts a new inner node in line 6 - 14, the function M has to be updated in order to meet the properties required in Definition 63. The map M has to be updated for each v_i along the path $v_r, \ldots, v_1$

and for all $s \in \mathrm{chr}(\sigma_1, \mathrm{vars}(v_r, \ldots, v_i))$ as follows:

$$M(v_i, s) := M(v_i, s) \cup \{v_{i+1}\}.$$

The nodes along the path $v_r, \ldots, v_2$ have to be updated analogously.

The function M is realized via a map and, therefore, insertion is bounded by $O(\log |\Sigma|)$ where $|\Sigma|$ is the number of signature symbols. As a result, updating the nodes along a path with length n, is bounded by the following complexity:

$$O(n * (|\,\mathrm{chr}(\sigma_1, \mathcal{W})| + |\,\mathrm{chr}(\sigma_2, \mathcal{W})|) * \log |\Sigma|).$$

In the context of clause sets from BSH-Y2, the characteristic functions of σ_1 and σ_2 have size at most two and the index has depth at most three.

The procedure EntryDelete (Algorithm 13) does not change the context tree when removing a term. Consequently, no update of M becomes necessary.

Theorem 64 (Correctness)**.**
Let ρ be a substitution, $T = (V, E, \mathrm{subst}, v_r)$ a context tree, and $FT = (V, E, \mathrm{subst}, v_r, M)$ be a filtered context tree. Then

$$v \in \mathrm{FilteredLookup}(FT, v_r, \rho, \mathrm{Test}) \;\Rightarrow\; v \in \mathrm{Lookup}(T, v_r, \rho, \mathrm{Test})$$

where Test *is one of the following test functions:* UnifyTest *(Algorithm 4),* GenTest *(Algorithm 6), and* InstTest *(Algorithm 8).*

Proof:
Since, the algorithm only restricts the number of nodes in the context tree which are considered for testing, the correctness follows from the correctness of Lookup (Theorem 38).

Theorem 65 (Completeness)**.** *Let ρ be a query substitution, $T = (V, E, \mathrm{subst}, v_r)$ a context tree, and $FT = (V, E, \mathrm{subst}, v_r, M.\,\mathrm{succ})$ be a filtered context tree. Then*

$$v \in \mathrm{FilteredLookup}(FT, v_r, \rho, \mathrm{Test}) \Leftarrow v \in \mathrm{Lookup}(T, v_r, \rho, \mathrm{Test}),$$

where Test *is one of the following test functions:* UnifyTest *(Algorithm 4),* GenTest *(Algorithm 6), and* InstTest *(Algorithm 8).*

Proof. Let ρ be a query substitution, $T = (V, E, \mathrm{subst}, v_r)$ a context tree, and $FT = (V, E, \mathrm{subst}, v_r, M.\,\mathrm{succ})$ be the respective filtered context tree. Assume the following holds:

$$v_n \in \mathrm{Lookup}(T, v_r, \rho, \mathrm{UnifyTest}).$$

The proof for GenTest and InstTest follows analogously. By soundness and correctness of Lookup, it follows that there is a path $v_r, v_1, \ldots, v_n$ from the root to the leaf v_n with $\tau_i = \mathrm{subst}(v_i)$, and

$$\exists\sigma\forall x(x\tau_1 \ldots \tau_n\sigma\rho = x\rho\sigma). \tag{5.1}$$

Because of condition 2 of context trees (Definition 36), it holds for all τ_i that

$$\exists\sigma\forall x(x\tau_i\sigma\rho = x\rho\sigma). \tag{5.2}$$

It remains to show that for all v_i on this path $v_{i+1} \in \text{succ}(v_i, \rho, \text{vars}(v_1, \ldots, v_i))$. The proof is by contradiction. Assume there is an i such that the following holds:

$$v_{i+1} \notin \text{succ}(v_i, \rho, \text{vars}(v_1 \ldots v_i)).$$

We have to distinguish two cases

- Assume there is a $x \in \text{dom}(\rho) \cap \text{vars}(v_1, \ldots, v_i) \cap \mathcal{W}$ with $\text{top}(x\rho) = f$ and $f \in \Sigma$. $x \in \text{vars}(v_1, \ldots, v_i)$ implies $x \in \text{vars}(\text{cod}(\tau_j))$ for some $j \in \{1, \ldots, i\}$ and $x \notin \text{dom}(\tau_k)$ for all $1 \leq k \leq j$.
  Because $x \in \mathcal{W}$ and condition 3 of context trees (Definition 36) it follows: there is a $l \in \{i+1, \ldots, n\}$ with $x \in \text{dom}(\tau_l)$.
  From 5.2 it follows that $\exists\sigma(x\tau_l\sigma\rho = x\rho\sigma)$ and, therefore, either (i) $\text{top}(x\tau_l) = f$ or (ii) $x\tau_l \in \mathcal{X}$.

  (i) From $\text{top}(x\tau_l) = f$, it follows that $f \in \text{chr}(\tau_l, \text{vars}(v_1, \ldots, v_{l-1}))$.
  Consequently, $v_{i+1} \in M(v_i, f)$ by definition of FT (Definition 63).

  (ii) From $x\tau_l \in \mathcal{X}$, it follows that $\bot \in \text{chr}(\tau_l, \text{vars}(v_1, \ldots, v_{l-1}))$.
  Consequently, $v_{i+1} \in M(v_i, \bot)$ by definition of FT (Definition 63).

  As a results form (i) and (ii), $v_{i+1} \in M(v_i, f) \cup M(v_i, \bot)$.
  By definition of succ (Definition 63) it follows

  $$v_{i+1} \in \text{succ}(v_i, \rho, \text{vars}(v_1 \ldots v_i))$$

  contradicting the assumption.

- Assume there is no $x \in \text{dom}(\rho) \cap \text{vars}(v_1, \ldots, v_i) \cap \mathcal{W}$ with $\text{top}(x\rho) = f$ and $f \in \Sigma$. If there is an $x \in \text{dom}(\rho)$ such that one of the following holds
  - $x \in \mathcal{X}$
  - $\text{top}(x\rho) \in \mathcal{X}$
  - $\text{top}(x\rho) \in \mathcal{U}$,

  then $\text{chr}(\rho, \text{vars}((v_1, \ldots, v_i)) = \{\bot\}$. By definition of succ (Definition 63) it follows

  $$v_{i+1} \in \text{succ}(v_i, \rho, \text{vars}(v_1 \ldots v_i))$$

  contradicting the assumption.
  If none of the above cases occur then we have for all $x \in \text{dom}(\rho)$ that $x \notin \text{vars}(v_1 \ldots v_i)$. Consequently, we have two distinguish the following two cases:
  - if $x \in \bigcup_{1 \leq j \leq i} \text{dom}(\tau_j)$ for all x then v_i is a leaf contradicting the fact that there is a v_{i+1} with $(v_i, v_{i+1}) \in E$.
  - if there is a x with $x \notin \bigcup_{1 \leq j \leq i} \text{vars}(\text{cod}(\tau_j))$ then ρ is not a query substitution for T.

□

5.3. Implementation in Spass–Y2

The index that is implemented in SPASS is the substitution tree index, which is a special kind of context trees. For this reason, I have integrated the filtering technique of filtered context trees into the substitution tree index of SPASS. The new version of SPASS that contains this new index and the reasoning procedures that I present in Chapter 6 and in Chapter 7 is called SPASS–Y2.

In SPASS, symbols are internally represented as integers. Consequently, they can be compared with respect to their integer value. So, the implementation of the map M uses CSB^+-trees [RR00] a cache conscious variant of B-trees.

The implementation of the set of variables of a path $\text{vars}(v_r = v_1, \ldots, v_n)$ is realized via a marking mechanism. Each time a substitution τ of a node is compatible with the current query ρ all index variables of $\text{dom}(\tau)$ are marked.

Because of the fact that one node of a filtered context tree could be reached via several symbols from its parent node, the retrieval algorithm marks each visited node in order to avoid multiple inspections of the same node.

Although working with every retrieval operation, I have implemented a separate version of the procedure FilteredLookup (Algorithm 14) for each of the retrieval operations: unification, instantiation, and generalization. This enables the procedure to exclude more subnodes of a given filtered context tree respecting the current retrieval operation. For example, assume the retrieval for instances of the substitution $\{w_i \mapsto g(x)\}$. In this case, nodes that solely contain substitutions of the form $\{w_i \mapsto x\}$ do not contribute and can be excluded from further processing. A similar argument holds for generalizations.

5.4. Further Improvements

There are further opportunities to improve the current implementation of the term index of SPASS–Y2. For example, the occurrence check for the unification operation can be omitted.

In the context of ontologies, the notion of function variables provides a mechanism to query for predicate symbols. For example, one can query the index for terms that contain the symbol a as its second argument. The respective query term is $F(x, a)$. Applying this query to the context tree given in Figure 5.1 returns the terms $f(c, a)$ and $h(d, a)$. For example, this feature could be used in order to answer queries like: "What is the relation of Angela Merkel and Hamburg".

We can also use context trees to index each term stored in the context tree by each of its symbols. For example, consider the term $f(c, a)$ which is stored in the context tree of Figure 5.1. Following the path from the root to the leaf we find the substitutions τ_1 and τ_2 with $f(c, a) = x_0\tau_1\tau_2$. The order of the application of the substitutions τ_1 and τ_2 to x_0 does not matter, because $x_0\tau_1\tau_2 = f(c, a) = x_0\tau_2\tau_1$. Using this property a filtered

context tree index can store both paths. Depending on the query one or the other is more efficient. Consider the query term $F(x, a)$. Here the only symbol occurring is a. In order to restrict the search space, the retrieval operation follows the path $\tau_1\tau_2$. If we consider, however, the query term $f(x, y)$ it is more efficient to first consider τ_2 because $f \in \mathrm{cod}(\tau_2)$. This increases the size of the tree exponentially. However, in the case of the YAGO ontology, this is affordable because a filtered context tree storing terms from BSH-Y2 has depth at most three. This approach provides a very efficient retrieval mechanism. A similar idea is used for the implementation of relational data base system, where an index is created for each of its arguments. For example, the tuple (a, b, c) can be obtained by querying the index of the first argument for a, querying the index of the second argument for b or querying the index of the third argument for c. An implementation of this can be found in [NW08].

5.5. Summary

Filtered context tree indexing is a powerful invention for storing large amounts of terms and efficiently performing queries on the index; in particular, for terms occurring in an ontology that contains several million clauses. When querying the index, the filtered context tree index enables the retrieval operations to avoid the inspection of unsuccessful subtrees of the context tree index. As a consequence, it performs a more goal oriented search. SPASS was not able to load YAGO into its index. Integrating the filtering technique of the filtered context tree index into the substitution tree index of SPASS–Y2, enables SPASS–Y2 to load YAGO into its index and also to efficiently perform reasoning tasks on this clause set. The presented filtering technique is sound and complete.

6. Superposition Calculus for BSH-Y2

In this chapter, I present the first superposition based reasoning calculus that efficiently decides the satisfiability of BSH-Y2 ontologies consisting of several million axioms. The set BSH-Y2 is a subset of the Bernays–Schönfinkel Horn fragment with equality. It is able to represent the YAGO ontology as well as large parts of the ontologies SUMO (SUMO-Y2) and CYC (CYC-Y2). In general, verifying the satisfiability in the Bernays–Schönfinkel Horn fragment is EXPTIME complete. Therefore, standard reasoning procedures are too prolific for reasoning in such large ontologies; the experiments in Chapter 8 confirm this.

Successful reasoning about the BSH-Y2 fragment requires a calculus that avoids the prolific behavior of the standard reasoning calculus. Examining the produced clauses of the standard first-order reasoning calculus, one can make two observations. First, ordered resolution produces too many clauses. Second, applying hyperresolution, instead of ordered resolution, is still too prolific because of the transitivity axioms of BSH-Y2. A solution for this problem is the chaining calculus which computes only that part of the transitive closure that is sufficient for completeness. The chaining calculus turned out to be effective for reasoning about the transitivity in BSH-Y2 ontologies. However, the chaining calculus requires ordered resolution for completeness.

In [SWW10], I have developed a calculus which uses hyperresolution together with the chaining calculus. This calculus is complete because an ordering restriction of the chaining calculus has been dropped. The resulting reasoning procedure saturates the YAGO ontology in less than one hour. However, this calculus is not able to saturate clause sets containing defined relations of BSH-Y2 in acceptable time. The reason for this observation is that a non-ground transitive atom that occurs in a defined relation causes the chaining calculus to inspect the whole transitive closure of this predicate. This problem arises already if one only adds the following clause to the YAGO ontology:

$$\mathrm{bornIn}(x, y), \mathrm{locatedIn}(y, z) \rightarrow \mathrm{bornInTr}(x, z).$$

In this chapter, I present my solution to the above problems: a *two-layered superposition calculus*. The two-layered superposition reasoning calculus separates the reasoning about non-transitive atoms from the reasoning about transitive atoms. This calculus combines hyperresolution and the chaining calculus, and it is at the same time effective, sound, complete, and terminating for BSH-Y2. I also provide the respective proofs in this chapter. As the experiments in Chapter 8 confirm, the new calculus, within the superposition reasoning framework, is an efficient saturation procedure for BSH-Y2 ontologies that consist of several million clauses.

Furthermore, if N is saturated with the new two-layered calculus, then it is an efficient representation of its minimal model. I use this observation in the definition of the efficient query answering procedure that I present in Chapter 7.

This chapter is structured as follows. First, it illustrates the problem of the chaining calculus that occurs if defined relations involving transitivity are present. Section 6.2 introduces the new two-layered superposition reasoning calculus for BSH-Y2. Section 6.3 proves the soundness, completeness, and termination of the calculus. Finally, Section 6.4 depicts details about the implementation of the new calculus in SPASS–Y2.

6.1. Saturation Strategy

The chaining calculus (Section 2.2) is defined in terms of an admissible ordering that may avoid the generation of the whole transitive closure. However, in the presence of a non-ground transitive atom, the ordering does not prevent the negative chaining rule from inspecting the whole transitive closure. This causes the generation of quadratically many new clauses. Consequently, the chaining calculus is not feasible for practical saturating a BSH-Y2 ontology consisting of several million clauses over several million constants. A possible solution is given via a selection function. But, the problem that the chaining calculus requires ordered resolution for completeness, still exists. The following example illustrates the problem of negative chaining if applied to a defined relation.

Assume a defined relation from BSH-Y2 containing the transitive predicate locatedIn as follows:

$$\text{bornIn}(x, y), \text{locatedIn}(y, z) \rightarrow \text{bornInTr}(x, z), \tag{6.1}$$

and the clause set:

$$\rightarrow \text{bornIn}(\text{AngelaMerkel}, \text{Hamburg}) \tag{6.2}$$

$$\rightarrow \text{locatedIn}(\text{Saarland}, \text{Germany}) \tag{6.3}$$
$$\rightarrow \text{locatedIn}(\text{Saarbrücken}, \text{Saarland}) \tag{6.4}$$
$$\rightarrow \text{locatedIn}(\text{Hamburg}, \text{Germany}) \tag{6.5}$$
$$\rightarrow \text{locatedIn}(\text{Germany}, \text{Europe}). \tag{6.6}$$

Further, assume the following precedence of constants:

$$\text{Saarbrücken} \succ \text{Saarland} \succ \text{Hamburg} \succ \text{Germany} \succ \text{Europe}\,.$$

The rule negative chaining applied to the transitive atom $\text{locatedIn}(y, z)$ of Clause 6.1 infers the transitive closure of locatedIn. All ordering requirements are fulfilled because the variables y, z and constants are incomparable and the atom $\text{locatedIn}(y, z)$ is strictly maximal in the clause 6.1. Consequently, negative chaining derives the below clauses from the clauses 6.3 – 6.6 and clause 6.1.

$$\text{bornIn}(x, \text{Saarland}), \text{locatedIn}(\text{Germany}, z) \rightarrow \text{bornInTr}(x, z) \quad (6.7)$$
$$\text{bornIn}(x, \text{Saarbrücken}), \text{locatedIn}(\text{Saarland}, z) \rightarrow \text{bornInTr}(x, z) \quad (6.8)$$
$$\text{bornIn}(x, \text{Hamburg}), \text{locatedIn}(\text{Germany}, z) \rightarrow \text{bornInTr}(x, z) \quad (6.9)$$
$$\text{bornIn}(x, \text{Germany}), \text{locatedIn}(\text{Europe}, z) \rightarrow \text{bornInTr}(x, z) \quad (6.10)$$
$$\text{bornIn}(x, \text{Saarbrücken}), \text{locatedIn}(\text{Germany}, z) \rightarrow \text{bornInTr}(x, z) \quad (6.11)$$
$$\text{bornIn}(x, \text{Saarland}), \text{locatedIn}(\text{Europe}, z) \rightarrow \text{bornInTr}(x, z) \quad (6.12)$$
$$\text{bornIn}(x, \text{Hamburg}), \text{locatedIn}(\text{Europe}, z) \rightarrow \text{bornInTr}(x, z) \quad (6.13)$$
$$\text{bornIn}(x, \text{Saarbrücken}), \text{locatedIn}(\text{Europe}, z) \rightarrow \text{bornInTr}(x, z) \quad (6.14)$$

In order to avoid the generation of the clauses 6.7–6.14, during the saturation process, one can select the literal $\text{bornIn}(x, y)$. As a result, negative chaining is not applicable anymore. An application of ordered resolution to the clause 6.1 and clause 6.2 derives the following clause:

$$\text{locatedIn}(\text{Hamburg}, z) \rightarrow \text{bornInTr}(\text{AngelaMerkel}, z), \quad (6.15)$$

where x and y are instantiated by constants. The variables of the atoms are not disjoint in the clause and, consequently, the atoms are comparable. If, by the choice of the precedence, the following holds:

$$\text{locatedIn}(\text{Hamburg}, z) \prec \text{bornInTr}(\text{AngelaMerkel}, z). \quad (6.16)$$

In this case, no further inferences are possible and the clause set is saturated. This can be achieved by defining a precedence on the predicates as follows: bornInTr > locatedIn. Such an ordering of the predicates is always possible because each defined relation in BSH-Y2 is acyclic (Section 4.1.2).

The following provides a definition of the respective selection function.

Definition 66 (Selection for BSH-Y2)**.**

$$\text{sel}(\Gamma \rightarrow \Delta) = \begin{cases} \{A\} & \text{, if there is } A \in \Gamma \text{ with } A \text{ non-transitive} \\ \emptyset & \text{, else} \end{cases}$$

The defined selection function guaranties that all inferences between non-transitive atoms are performed first. The superposition calculus is complete with this selection function [BG98, BG01].

Note, using a reasoning calculus with *ordering constraints*, as suggested in [NR01], does not prevent the generation of the clauses 6.7 – 6.14 because variables and constants are not comparable. For example, when generating the clause 6.7, the clause with constraint, as imposed by negative chaining, is the following:

$$\text{bornIn}(x, \text{Saarland}), \text{locatedIn}(\text{Germany}, z) \rightarrow \text{bornInTr}(x, z) \mid z \not\approx \text{Saarland}\,. \quad (6.17)$$

Because of the fact that the variable z and the constant Saarland are not comparable, the constraint is trivially fulfilled as long as the variable z is not instantiated. As a consequence, the constraint of clause 6.17 does not prevent negative chaining from deriving the following clause from the ordered constrained clause 6.17 and the clause 6.6:

$$\text{bornIn}(x, \text{Saarland}), \text{locatedIn}(\text{Europe}, z) \rightarrow \text{bornInTr}(x, z) \mid \begin{array}{l} z \not\succeq \text{Saarland} \\ z \not\succeq \text{Gemany} \end{array} \quad (6.18)$$

Instead of avoiding the inferences, the ordered constrained calculus only adds a further constraint ($z \not\succeq$ Germany) which is again trivially fulfilled if the variable z is not instantiated. So, reasoning with ordered constraints does not prevent the generation of the clauses 6.7 – 6.14, and, consequently, reasoning with ordered constraints does not prevent the negative chaining rule from processing the whole transitive closure.

6.2. Superposition Calculus for BSH-Y2

This section introduces the two-layered superposition reasoning calculus for BSH-Y2. The calculus performs a reasoning on two layers that separate the reasoning about transitive predicates from reasoning about non-transitive predicates. Likewise, in the standard superposition framework, reasoning about sorts can be separated from reasoning about non sort clauses. Reasoning about sorts independently is much more effective via special data structures and reasoning procedures [Wei01].

The separation of reasoning about transitive predicates from reasoning about non-transitive predicates, allows the calculus to use the prolific ordered resolution rule only for reasoning about the transitive predicates where it is required for completeness. In all other cases, hyperresolution is applied which is much less prolific than ordered resolution. The two-layered calculus reasons about sorts independently from the two layers like in the standard reasoning calculus. Figure 6.1 illustrates the separation of the clauses of a reasoning problem.

During the saturation of a clause set N, each clause is assigned to exactly one layer or to the sort theory. Each layer has its own calculus rules which are applied only to the clauses of this layer. The sort theory contains the static sort theory S_N of N. The respective sort reasoning rules, EmS, SSi, and SST, are applied to the clauses of both the transitive layer and the non-transitive layer. The non-transitive reasoning layer applies hyperresolution, HyperY2, and object equality cutting, OECut, to its clauses, and the transitive reasoning layer applies the chaining rules, OChainY2 and NChainY2, and ordered resolution, OReY2, to its clauses.

In order to use the standard superposition framework and its sophisticated implementation in SPASS, I have developed the new calculus in such a way that the clause set is not explicitly split into the different layers. Instead, the separation is done by each individual calculus rule; similar to the sort reasoning calculus. This means that each

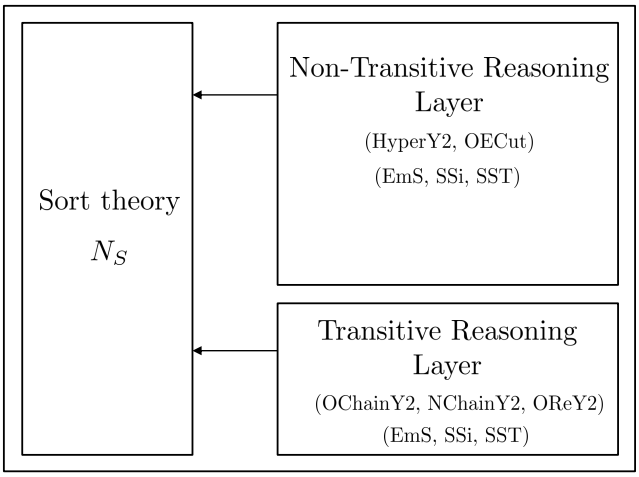

Figure 6.1.: Reasoning Layers

inference rule can be either applied to a clause that contains at least one non-transitive predicate or to a clause that contains only transitive predicates. The resulting layered superposition calculus is called $\mathcal{C}^{\succ}_{\mathrm{Tr}}$ and defined in the following.

Definition 67 (Two-Layered superposition reasoning calculus $\mathcal{C}^{\succ}_{\mathrm{Tr}}$)**.** *The two-layered superposition reasoning calculus $\mathcal{C}^{\succ}_{\mathrm{Tr}}$ is defined with respect to (i) a set of clauses N from BSH-Y2 not containing transitivity axioms, (ii) the respective sort theory S_N, (iii) the set* Tr *containing the transitive predicates of N, and (iv) an admissible ordering $\succ$.*

The calculus $\mathcal{C}^{\succ}_{\mathrm{Tr}}$ consists of the following inferences defined in the remainder of this chapter: (i) HyperY2, (ii) OECut, (iii) OChainY2, (iv) NChainY2, (v) OReY2, (vi) EmS, (vii) SSi, and (viii) SST. The calculus $\mathcal{C}^{\succ}_{\mathrm{Tr}}$ uses subsumption deletion and tautology deletion as the redundancy criteria during the saturation for each layer.

6.2.1. Non-Transitive Reasoning Layer

The calculus of the non-transitive layer consists of two rules. The first is an instance of the hyperresolution rule (Section 2.2) and the second rule is object equality cutting which ensures that the unique name assumption is respected.

Ordered Hyperresolution for BSH-Y2 (HyperY2)

$$\frac{(1 \leq i \leq n) \quad \Theta_i \,\|\, \Gamma_i \to A_i \quad \Theta \,\|\, T_1, \ldots, T_m, B_1, \ldots, B_n \to \Delta}{(\Theta, \Theta_1, \ldots, \Theta_n \,\|\, T_1, \ldots, T_m, \Gamma_1, \ldots, \Gamma_n \to \Delta)\sigma},$$

where $n \geq 1$, $T_1, \ldots, T_m$ are transitive atoms, $\Theta_1, \ldots, \Theta_n, \Theta$ are solved, $\Gamma_1, \ldots, \Gamma_n$ contain only transitive atoms, $B_1, \ldots, B_n$ are non-transitive atoms, σ is the simultaneous

most general unifier of A_i and B_i for all $i \in \{1, \dots n\}$, respectively, and $A_i\sigma$ are strictly maximal in $(\Theta_i \parallel \Gamma_i \to A_i)\sigma$.

This instance of ordered hyperresolution simulates several ordered resolution steps with respect to the selection function sel (Definition 66) without generating the intermediate results. This rule ignores all transitive atoms and is only applied to clauses containing at least one non-transitive atom. Consequently, this rule performs a complete saturation of the non-transitive reasoning layer. The rule HyperY2 derives clauses that have only transitive antecedents. Consequently, the derived clause belongs to the transitive layer if $\Delta = \emptyset$ or Δ contains a transitive atom.

Object Equality Cutting (OECut)

$$\frac{\parallel \to a \approx b}{\square},$$

where a and b are two different constants.

By the definition of BSH-Y2 in Section 4.1.3, equations occur only on the non-transitive reasoning layer. Consequently, the rule object equality cutting [SB05] is only considered for clauses of this layer. The rule ensures for a saturated clause set N of BSH-Y2 that the minimal model of the saturated clause set respects the unique name assumption, i.e., $N_I \models \mathrm{N}_{\mathrm{una}}$. Consequently, it is not necessary for a BSH-Y2 ontology to explicitly contain quadratically many disequations ($\{a \not\approx b \parallel a \neq b, a \in \mathcal{F}, b \in \mathcal{F}\}$).

6.2.2. Transitive Reasoning Layer

This section presents the reasoning calculus for the transitive reasoning layer. The calculus is an instance of the chaining calculus [BG98] which has also been shown in Section 2.2. The transitive layer consists of purely transitive clauses $\Theta \parallel \Gamma \to \Delta$, i.e. for all atoms $Q(t_1, t_2) \in \Gamma \cup \Delta$ it holds $Q \in \mathrm{Tr}$. The calculus presented in this section is the standard chaining calculus defined only for clauses from the transitive reasoning layer. It consists of the respective instances of ordered chaining, negative chaining and ordered resolution. Because all defined transitive predicates are independent from other transitive predicates, in the definition of the following rules, the antecedent of a clause is empty if the succedent is non empty.

Ordered Chaining for BSH-Y2 (OChainY2)

$$\frac{\Theta_1 \parallel \to Q(l, s) \quad \Theta_2 \parallel \to Q(t, r)}{(\Theta_1, \Theta_2 \parallel \to Q(l, r))\sigma}$$

where $Q \in \mathrm{Tr}$ is a transitive predicate, σ is the most general unifier of s and t, Θ_1 and Θ_2 are solved, $Q(t, r)\sigma$ is strictly maximal in $(\Theta \parallel \Gamma \to Q(t, r))\sigma$, $l\sigma \not\geq s\sigma$, $r\sigma \not\geq t\sigma$, and there are only transitive literals in Γ.

Negative Chaining for BSH-Y2 (NChainY2)

$$\frac{\Theta_1 \parallel \rightarrow Q(l,s) \quad \Theta_2 \parallel \Gamma, Q(t,r) \rightarrow}{(\Theta_1\Theta_2 \parallel \Gamma, Q(s,r) \rightarrow)\sigma}$$

where $Q \in \mathrm{Tr}$ is a transitive predicate, σ is the most general unifier of l and t, Θ_1 and Θ_2 are solved, $s\sigma \not\succeq l\sigma$, $r\sigma \not\succeq t\sigma$, $Q(t,r)\sigma$ is maximal with respect to $(\Theta \parallel \Gamma, Q(t,r) \rightarrow)\sigma$, and there are only transitive literals in Γ.

$$\frac{\Theta_1 \parallel \rightarrow Q(l,s) \quad \Theta_2 \parallel \Gamma, Q(t,r) \rightarrow}{(\Theta_1, \Theta_2 \parallel \Gamma, Q(t,l) \rightarrow)\sigma}$$

where $Q \in \mathrm{Tr}$ is a transitive predicate, σ is the most general unifier of s and r, Θ_1 and Θ_2 are solved, $l\sigma \not\succeq s\sigma$, $t\sigma \not\succ r\sigma$, $Q(t,r)\sigma$ is maximal with respect to $(\Theta \parallel \Gamma, Q(t,r) \rightarrow)\sigma$, and there are only transitive literals in Γ.

Ordered Resolution for BSH-Y2 (OReY2)

$$\frac{\Theta_1 \parallel \rightarrow Q(t_1,t_2) \quad \Theta_2 \parallel \Gamma, Q(s_1,s_2) \rightarrow}{(\Theta_1, \Theta_2 \parallel \Gamma \rightarrow)\sigma},$$

where $Q \in \mathrm{Tr}$ is a transitive predicate, σ is the most general unifier of $Q(t_1,t_2)$ and $Q(s_1,s_2)$, Θ_1 and Θ_2 are solved, $Q(s_1,s_2)\sigma$ is strictly maximal in $(\Theta \parallel \Gamma, Q(s_1,s_2) \rightarrow)\sigma$, and there are only transitive literals in Γ.

6.2.3. Sort Reasoning

The sort theory S_N of a BSH-Y2 ontology is static (Section 4.1.3) because it only consists of facts ($\parallel S(a) \rightarrow$) and subsort relations ($S_1(x) \parallel \rightarrow S_2(x)$). As a consequence, $N_I \models S(a)$ iff $\mathrm{S}_N \models S(a)$. This property is invariant on the saturation of N, while fixing S_N from the beginning, because S_N only consists of facts and subsort relations [GMW97].

Hence, when deriving a clause $S(a), \Theta \parallel \Gamma \rightarrow \Delta$ with $\mathrm{S}_N \not\models S(a)$, the clause is a tautology and can be deleted. This observation is used for the implementation of $\mathcal{C}^{\succ}_{\mathrm{Tr}}$. Note, the relations $\mathrm{S}_N \models S(a)$ and $\mathrm{S}_N \models \exists x\, S_1(x) \wedge \ldots \wedge S_n(x)$ can be efficiently decided by specific algorithms [Wei01]. The sort theory S_N is acyclic by definition of BSH-Y2. Reasoning about sort theories with these properties can be performed by an efficient implementation of the two rules *Empty sort* and *Sort simplification* [Wei01, GMW97]. The presented calculus is an instance of the general sort reasoning calculus (Section 2.2) that makes use of the fact that the sort theory of a BSH-Y2 ontology is static. This enabled me to integrate the sort reasoning procedures into the implementation of the inference rules of the two-layered calculus (Section 6.4) yielding a more efficient reasoning procedure for BSH-Y2.

Empty Sort

$$\frac{S(x), \Theta \parallel \Gamma \to \Delta}{(\Theta \parallel \Gamma \to \Delta)\sigma},$$

if σ is a substitution with $S(x\sigma)$ is ground, $x \notin \text{vars}(\Gamma \cup \Delta)$, and $S_N \models S(x\sigma)$.

Sort Simplification

$$\mathcal{R}\frac{S(a), \Theta \parallel \Gamma \to \Delta}{\Theta \parallel \Gamma \to \Delta},$$

if $\mathrm{S}_N \models S(a)$. In the sort theory of a clause set from the BSH-Y2 sort simplification coincides with sort resolution.

Static Soft Typing

$$\mathcal{R}\frac{S(x), \Theta \parallel \Gamma \to \Delta}{},$$

if $S_N \not\models \exists x\, S(x)$.

6.3. Soundness, Termination, and Completeness

The two-layered superposition reasoning calculus for the BSH-Y2 is sound, complete, and terminating. This chapter presents the respective proofs.

6.3.1. Soundness

Theorem 68 (Soundness)**.** *The two layered reasoning calculus is sound.*

Proof. The soundness of the calculus is implied by the soundness of each individual rule. The superposition calculus $\mathcal{C}^{\succ}_{\mathrm{Tr}}$ is composed of instances of rules from the standard first-order reasoning framework (Section 2.2) which are known to be sound [Wei01, BG01]. □

6.3.2. Completeness

Lemma 69. *If N is a saturation, with respect to $\mathcal{C}^{\succ}_{\mathrm{Tr}}$, of a clause set consisting of clauses from BSH-Y2 and $\square \notin N$ then for all ground terms $S(a)$ the following holds:*

$$N_I \models S(a) \Leftrightarrow \mathrm{S}_N \models S(a).$$

Proof. For the static sort theory S_N of N from the BSH-Y2 it holds that $\mathrm{S}_N \subseteq N$ because the only clauses of N where a sort atom occurs positively is either a subsort relation $(S_1(x) \parallel \rightarrow S_2(x))$ or a sort fact $(\parallel \rightarrow S(a))$. In addition, there are no cycles produced by the subsort relations of N. Then from [GMW97, SS89] the lemma holds. □

Lemma 70. *If N is the saturation, with respect to $\mathcal{C}^{\succ}_{\mathrm{Tr}}$, of a clause set consisting of clauses from the BSH-Y2, $\square \notin N$, and N_I the minimal model of N. A ground atom A is produced in N_I by a ground instance $\Theta \parallel \Gamma \rightarrow Q(s,t)$ of a clause N with Γ contains only transitive atoms.*

Proof. The proof is by contradiction. Assume a ground instance as follows:

$$C\sigma = (\Theta \parallel \Gamma, B_1, \ldots, B_n \rightarrow A)\sigma$$

that produces the atom $A\sigma$ in N_I and $B_1, \ldots, B_n$ are non-transitive atoms. Assume $C\sigma$ is the smallest productive clause that contains non-transitive antecedents. By definition of N_I it follows:

$$\{B_1\sigma, \ldots, B_n\sigma\} \subseteq R^*_C$$

Since, $C\sigma$ is the smallest clause, all $B_i\sigma$ have been produced by ground instances $\Theta_i \parallel \Gamma_i \rightarrow B_i\sigma$ with $B_i \succ \Gamma_i$ for $i \in \{1, \ldots, n\}$. An application of Hyperresolution derives a smaller clause that produces $A\sigma$. This contradicts the assumption that $C\sigma$ produces $A\sigma$. □

Lemma 71. *If N is the saturation, with respect to $\mathcal{C}^{\succ}_{\mathrm{Tr}}$, of a clause set consisting of clauses from the BSH-Y2 and $\square \notin N$. A ground atom $Q(s,t)$ with $Q \in \mathrm{Tr}$ is produced by a clause $\Theta \parallel \rightarrow Q(s,t)$ in N_I.*

Proof. Assume $Q(s,t)$ has been produced by a ground instance of a clause $\Theta \parallel \Gamma \rightarrow Q(s,t)$ with $\Gamma \neq \emptyset$. Then Γ does not contain transitive predicates because Q is transitive independent. All atoms $A_i \in \Gamma$ have been produced by clauses $\Theta_i \parallel \Gamma_i \rightarrow A_i$ with $A_i \succ \Gamma_i$. An application of Hyperresolution derives a smaller clause producing $Q(s,t)$. As a result, the clause $\Theta \parallel \Gamma \rightarrow Q(s,t)$ cannot be productive. □

Lemma 72. *Let N be the saturation, with respect to $\mathcal{C}^{\succ}_{\mathrm{Tr}}$, of a clause set consisting of clauses from BSH-Y2 using the selection* sel *(Definition 66) and $\square \notin N$ then for all clauses $C \in N$ we have that $N_I \models C$.*

Proof. For the sort theory $\mathrm{S}_N \subseteq N$ we have that $N_I \models S_N$ as a consequence form Lemma 69. The following proof by contradiction shows that for all other clauses $C \in N$ and all ground instances $C\sigma$ it holds that $N_I \models C\sigma$. Assume a ground instance $C\sigma$ of a clause $C = \Theta \,\|\, \Gamma \to \Delta$ in N such that $N_I \not\models C\sigma$. Assume further, that $C\sigma$ is the smallest ground instance of a clause in N with this property. Consequently, $C\sigma$ is not productive and we have to distinguish the following cases.

(a) If C has an unsolved constraint then either $C = S(x), \Theta' \,\|\, \Gamma \to \Delta$ and $S(x)$ with $x \notin \mathrm{vars}(\Gamma \cup \Delta)$ or $C = S(a), \Theta' \,\|\, \Gamma \to \Delta$. In the first case, by assumption $N_I \models S(x)\sigma$ which is equivalent to $\mathrm{S}_N \models S(x)\sigma$ (Lemma 69). Consequently, empty sort derives a smaller clause D with $N_I \not\models D\sigma$ which contradicts the fact that $C\sigma$ is minimal. In the second case, by assumption $N_I \models S(a)$ and an application of sort simplification reduces C to a smaller clause. Both cases contradict the assumption that C is the minimal non productive clause.

(b) If the clause C has the following form $C = \Gamma, B_1, \ldots, B_m \to \Delta$ with Γ a possibly empty set of transitive atoms, B_j are non-transitive atoms and either $\Delta = \emptyset$ or $\Delta = \{A\}$. By the selection function sel the atoms B_j are selected and therefore, $C\sigma$ is not productive. Then by construction of N_I (Definition 56) and the monotonicity of the construction (Definition 57) we have that

$$\{T_1\sigma, \ldots, T_n\sigma, B_1\sigma, \ldots, B_m\sigma\} \subseteq R^*_C \text{ and, if } \Delta = \{A\} \text{ then } A\sigma \notin R^*_C.$$

Consequently, the ground atoms $B_i\sigma$ have been produced by ground instances $(\Theta_i \,\|\, \Gamma_i \to B_i)\sigma$ of clauses from N. As a result, $\Gamma_i \prec B_i\sigma$ and Γ_i, by Lemma 70, contains only transitive atoms. This enables a hyperresolution step deriving a smaller clause

$$T_1\sigma, \ldots, T_n\sigma, \Gamma_1\sigma, \ldots, \Gamma_m\sigma_m \to \Delta\sigma$$

Consequently, C is not the minimal non productive clause.

(c) If C is of the form $C = T_1, \ldots, T_n \to$ then by the construction of N_I (Definition 56) and the monotonicity of the construction (Definition 57) we get

$$\{T_1\sigma, \ldots, T_n\sigma\} \subseteq R^*_C$$

There is a $T\sigma \in \{T_1\sigma, \ldots, T_n\sigma\}$ such that $T\sigma$ is maximal in $C\sigma$. We have to distinguish two cases either $T\sigma$ has been produced or there is a rewrite proof for $T\sigma$ in R^*_C.

(i) Assume $T_i\sigma$ has been produced. By Lemma 71, it has been produced by a ground unit clause $\to T_i\sigma$. An application of ordered resolution for transitive relations is possible deriving a smaller clause and C is not the minimal non productive clause.

(ii) If $T_i\sigma$ has not been produced then there is a rewrite proof $l \Downarrow_Q^{R_C} r$ with $T_i\sigma = Q(l, r)$. The rewrite proof is a chain with $l = t_1$ and $r = t_n$

$$Q(t_1, t_2), \ldots, Q(t_{n-1}, t_n)$$

with the following ordering

$$t_1 \succ \cdots \succ t_i \prec \cdots \prec t_n$$

Each $Q(t_i, t_{i+1})$ has been produced by a ground unit clause (Lemma 71) $D =\!\!\rightarrow Q(t_i, t_{i+1})$. Now, we have to distinguish two cases either $l \succ r$ or $l \preceq r$. First, assume $l \succ r$. Since we have that $t_1 \succ t_2$ in the above chain a negative chaining step is applicable between $C\sigma$ and D producing a smaller clause than C which is false in N_I. In the other case, if $l \preceq r$ then an application of negative chaining with $\rightarrow Q(t_{n-1}, t_n)$ derives a smaller clause and C is not the minimal non productive clause.

□

Lemma 73. *If N is a saturation, with respect to $\mathcal{C}^{\succ}_{\mathrm{Tr}}$, of a clause set from BSH-Y2 , $\square \notin N$ then $N_I \models \mathcal{A}_{\mathrm{Tr}}$.*

Proof. By Lemma 47 we have to show that all peaks commute in N_I. The following proves by induction on the clause order that all peaks in N_I commute. Assume a ground instance C of a clause from N which produces the atom $Q(l, r)$. By induction all peaks of R^*_C commute. So, it remains to show that all peaks of $(R_C \cup E_C)^*$ commute, too.

A peak of $(R_C \cup E_C)^*$ which is not in R^*_C has the following form $l' \Leftarrow t \Rightarrow r'$. Consequently, there are ground instances $\Gamma \rightarrow Q(l', t)$ and $\Gamma_2 \rightarrow Q(t, r')$ of clause from N which produce $Q(l', t)$ and $Q(t, r')$, respectively. W.l.o.g. assume C is the larger of these two. Because of the facts that $Q(l', t) \succ \Gamma_1$ and $Q(t, r') \succ \Gamma_2$ an ordered chaining application derives the clause $C' = \Gamma_1, \Gamma_2 \rightarrow Q(l', r')$. The ordering $\succ$ is admissible and, therefore, $C \succ C'$. Then we get $C' \in R^*_C$ from Lemma 72. As a result, the peak $l' \Leftarrow t \Rightarrow r'$ commutes in $(R_C \cup E_C)^*$. □

Lemma 74. *If N is the saturation, with respect to $\mathcal{C}^{\succ}_{\mathrm{Tr}}$, of a clause set consisting of clauses from BSH-Y2 and $\square \notin N$ then $N_I \models a \not\approx b$ for $a \neq b$.*

Proof. We show by contradiction that $N_I \not\models a \approx b$ for $a \neq b$. So, assume $a \approx b$ is the smallest equational atom with $N_I \models a \approx b$ for $a \neq b$. Consequently, there is a ground instance $C\sigma = \| \Gamma \rightarrow a \approx b$ of a clause $C \in N$ that produces $a \approx b$.

First, consider the case that, $\Gamma = \emptyset$, therefore, $C\sigma = \| \rightarrow a \approx b$ and an application of OECut derives $\square$ contradicting the fact that N is saturated and $\square \notin N$.

Second, consider the case $\Gamma \neq \emptyset$ where $\Gamma = B_1, B_2$ with B_1, B_2 are non defined binary predicates by definition of the BSH-Y2. The ground clause $C\sigma$ is productive by assumption and, consequently, $\Gamma\sigma \subseteq R^*_C$ and $a \approx b \notin R^*_C$. So, for B_1, B_2 there are ground instances $\rightarrow B_1\sigma$ and $\rightarrow B_2\sigma$ of clauses from N that have produced $B_1\sigma$ and $B_2\sigma$ in R^*_C, respectively. A hyperresolution application derives from these clauses and $C\sigma$ the clause

$$\| \rightarrow a \approx b$$

This contradicts the assumption that N is saturated and $C\sigma$ is productive. □

Theorem 75 (Completeness)**.** *If N is the saturation, with respect to $\mathcal{C}^{\succ}_{\mathrm{Tr}}$, of a clause set consisting of clauses from BSH-Y2 then $\square \in N$ if and only if N is unsatisfiable. If N is satisfiable then $N_I \models N \cup \mathcal{A}_{\mathrm{Tr}} \cup \mathrm{N}_{\mathrm{una}}$.*

Proof. If N contains a contradiction, namely $\square \in N$, then N is unsatisfiable. If $\square \notin N$ then from Lemma 72, Lemma 73 and Lemma 74 it follows $N_I \models N \cup \mathcal{A}_{\mathrm{Tr}} \cup \mathrm{N}_{\mathrm{una}}$. $\square$

6.3.3. Termination

Definition 76. *Let N be a clause set from the BSH-Y2, $\Sigma = (\mathcal{F}, \mathcal{R})$ the respective signature, and $\mathrm{Tr} \subseteq \mathcal{R}$ the set of transitive predicates. All clauses from N are acyclic by Definition of BSH-Y2. Consequently, a total ordering $>_{\mathcal{R}}$ on the predicates in $\mathcal{R}$ can be defined as follows.*

- *for all $T \in \mathrm{Tr}$ and for all $P \in \mathcal{R} \setminus \mathrm{Tr}$ it holds $P >_{\mathcal{R}} T$.*
- *Assume for each clause $\Theta \,\|\, \Gamma_{\mathrm{Tr}}, \Gamma \to A$ in N, with Γ_{Tr} is a set of transitive atoms and Γ a set of non-transitive atoms, the following ordering on the predicates of $\mathcal{R}$: For all $B \in \Gamma$ it holds that $\mathrm{top}(B) >_{\mathcal{R}} \mathrm{top}(A)$.*
- *The ordering can be extended to an ordering total on $\mathcal{R}$.*

For each clause $C = \Theta \,\|\, \Gamma_{\mathrm{Tr}}, \Gamma \to \Delta$ in N the order $\mathrm{ord}(C) \in \mathcal{R} \times \mathbb{N}$ is defined as

$$\mathrm{ord}(C) = (P, |\Gamma_{\mathrm{Tr}}| + |\Gamma|) \tag{6.19}$$

with P is maximal in $\mathrm{preds}(C)$ w.r.t. $>_{\mathcal{R}}$. The ordering $\succ_{\mathcal{C}}$ is the lexicographic extension of $>_{\mathcal{R}}$ and $>$:

$$(P_1, m_1) \succ_{\mathcal{C}} (P_2, m_2) \tag{6.20}$$

if and only if

1. *$P_1 >_{\mathcal{R}} P_2$ or*
2. *$P_1 = P_2$ and $m_1 > m_2$*

Since, $>_{\mathcal{R}}$ is total and well-founded on $\mathcal{R}$ and $>$ is total and well-founded on $\mathbb{N}$, also their lexicographic extension is total and well-founded on $\mathcal{R} \times \mathbb{N}$.

Lemma 77. *Let N be a clause set from the BSH-Y2, $\Sigma = (\mathcal{F}, \mathcal{R})$ the respective signature and $\mathrm{Tr} \subseteq \mathcal{R}$ the set of transitive predicates of N. For each*

$$(P, m) \in \mathcal{R} \times \mathbb{N} \tag{6.21}$$

the following set is finite up to subsumption

$$\{C \mid \mathrm{ord}(C) = (P, m) \wedge C \text{ is a Horn clause over } \Sigma\} \tag{6.22}$$

Proof. Assume a Horn clause $C = \Theta \parallel \Gamma \rightarrow \Delta$ with $\mathrm{ord}(C) = (P, m)$. By definition of $\mathrm{ord}(C)$, it follows that $|\Gamma| = m$. The clause C is Horn and, consequently, $|\Delta| \leq 1$. As a result, the number of non-sort atoms in C is smaller or equal than $m + 1$, i.e., $|\Gamma \cup \Delta| \leq m + 1$. Because of the fact that C is from BSH-Y2, all function symbols occurring in C are constants, and $\mathcal{F}$ and $\mathcal{R}$ are finite. As a result, the following set of atoms over Σ is finite up to variable renaming:

$$\mathcal{A} = \{P(s,t) \mid P \in \mathcal{R} \ \wedge \ s,t \in \mathcal{T}(\mathcal{F},\mathcal{X})\} \tag{6.23}$$

So, there are only finitely many possibilities for the set of atoms $\Gamma \cup \Delta$ with $|\Gamma \cup \Delta| \leq m + 1$.

It remains to show that there are only finitely many sort atoms over Σ. This means, the following set is finite up to variable renaming:

$$\{S(t) \mid S \in \mathcal{R} \ \wedge \ t \in \mathcal{T}(\mathcal{F},\mathcal{X})\} \tag{6.24}$$

This is finite up to variable renaming because all function symbols occurring in C are constants, and $\mathcal{F}$ and $\mathcal{R}$ are finite. Because of the fact that $|\Gamma \cup \Delta| \leq m + 1$ also the number of different variables is restricted. As a result, the number of different variables which need to be considered in Θ is bound. As a consequence, the set

$$\{C \mid \mathrm{ord}(C) = (P, m) \wedge C \text{ is a Horn clause over } \Sigma\} \tag{6.25}$$

is finite up to variable renaming.

□

Theorem 78 (Termination)**.** *If N is a clause set from BSH-Y2 then $\mathcal{C}_{\mathrm{Tr}}^{\succ}$ is terminating on N.*

Proof. Assume a clause set N from BSH-Y2, the respective signature Σ, and the ordering $\succ_{\mathcal{C}}$. This proof shows that the inference rules HyperY2, OChainY2, NChainY2, and OReY2 derive a clause that is either smaller than all premises with respect to $\succ_{\mathcal{C}}$, or that is smaller than or equal to the largest premise. If a rule derives a clause that is only smaller than or equal to the largest premise with respect to $\succ_{\mathcal{C}}$ then the proof shows that there are only finitely many equal clauses with respect to $\succ_{\mathcal{C}}$.

- HyperY2: Assume a clause

  $$D = \Theta \parallel T_1, \ldots, T_m, B_1, \ldots, B_n \rightarrow \Delta$$

  with T_i are transitive atoms and B_i non-transitive atoms. Assume w.l.g. B_1 is maximal among the B_i with respect to $>_{\mathcal{R}}$ and $i \in \{1, \ldots, n\}$. By definition of $\succ_{\mathcal{C}}$ also $\mathrm{top}(B_1) \succ_{\mathcal{R}} \mathrm{top}(A)$ if $\Delta = \{A\}$, and, consequently, $\mathrm{ord}(C) = (\mathrm{top}(B_1), m + n)$. An application of HyperY2 derives the clause

  $$C = \Theta, \Theta_1, \ldots, \Theta_n \parallel T_1, \ldots, T_m, \Gamma_1, \ldots, \Gamma_n \rightarrow \Delta$$

  from D and $D_i = \Theta_i \parallel \Gamma_i \rightarrow B_i$. Depending on Δ the following cases have to be considered:

(i) if $\Delta = \{A\}$ then $\mathrm{ord}(C) = (\mathrm{top}(A), m + |\Gamma_1| + \cdots + |\Gamma_n|)$.

(ii) if $\Delta = \emptyset$ then $\mathrm{ord}(C) = (\mathrm{top}(T), m + |\Gamma_1| + \cdots + |\Gamma_n|)$ for some $T \in T_1, \ldots, T_m, \Gamma_1, \ldots, \Gamma_n$.

All atoms of Γ_i are transitive and $\mathrm{ord}(D_i) = (B_i, |\Gamma_i|)$. Because $B_i >_{\mathcal{R}} A$ and $B_i >_{\mathcal{R}} T$ for all $T \in \mathrm{Tr}$, by Definition 76, $\mathrm{ord}(D_i) \succ_{\mathcal{C}} \mathrm{ord}(C)$ and $\mathrm{ord}(D) \succ_{\mathcal{C}} \mathrm{ord}(C)$. In summary, the clause C is smaller than all premises $D_1, \ldots, D_n$, and D.

- OChainY2: For a clause $D = \Theta_2 \parallel \rightarrow Q(t, r)$, OChainY2 derives a clause $C = \Theta_1, \Theta_2 \parallel \rightarrow Q(l, r)$ with $\mathrm{ord}(D) = \mathrm{ord}(C) = (Q, 0)$. By Lemma 77 the set

$$\{C \mid \mathrm{ord}(C) = (P, m) \wedge C \text{ is a Horn clause over } \Sigma\} \tag{6.26}$$

is finite up to subsumption. Consequently, OChainY2 can derive only finitely many clauses up to subsumption.

- NChainY2: For a clause $D = \Theta_2 \parallel \Gamma, Q(t, r) \rightarrow$, NChainY2 derives a clause $C = (\Theta_1, \Theta_2 \parallel \Gamma, Q(t, r) \rightarrow)\sigma$ with $\mathrm{ord}(D) = \mathrm{ord}(C)$. By Lemma 77 the set

$$\{C \mid \mathrm{ord}(C) = (P, m) \wedge C \text{ is a Horn clause over } \Sigma\} \tag{6.27}$$

is finite up to subsumption. Consequently, NChainY2 can derive only finitely many clauses up to subsumption.

- OReY2: For the clauses $D_1 = \Theta_1 \parallel \rightarrow Q(t_1, t_2)$ and $D_2 = \Theta_2 \parallel \Gamma, Q(s_1, s_2) \rightarrow$, OReY2 derives a clause $C = \Theta_1, \Theta_2 \parallel \Gamma \rightarrow$. We have to distinguish three cases

  1. if $\mathrm{ord}(D_2) = (Q, m)$ and $\mathrm{ord}(C) = (Q', m - 1)$ with $Q >_{\mathcal{R}} Q'$ than Q is the maximal predicate in D_2 w.r.t. $>_{\mathcal{R}}$. As a result, $\mathrm{ord}(D_1) \succ_{\mathcal{C}} \mathrm{ord}(C)$ and $\mathrm{ord}(D_2) \succ_{\mathcal{C}} \mathrm{ord}(C)$. Consequently, C is smaller than all premises: D_1 and D_2.

  2. if $\mathrm{ord}(D_2) = (Q, m)$ and $\mathrm{ord}(C) = (Q, m - 1)$
     then $\mathrm{ord}(D_2) \succ_{\mathcal{C}} \mathrm{ord}(C)$,
     but $\mathrm{ord}(D_1) \not\succ_{\mathcal{C}} \mathrm{ord}(C)$.
     By Lemma 77 the following holds:

$$\{C \mid \mathrm{ord}(C) = (P, m - 1) \wedge C \text{ is a Horn clause over } \Sigma\} \tag{6.28}$$

     is finite up to subsumption.

  3. if $\mathrm{ord}(D_2) = (Q', m)$ with $Q' >_{\mathcal{R}} Q$
     then $\mathrm{ord}(C) = (Q', m - 1)$ and $\mathrm{ord}(D_2) \succ_{\mathcal{C}} \mathrm{ord}(C)$,
     but $\mathrm{ord}(D_1) \not\succ_{\mathcal{C}} \mathrm{ord}(C)$.
     By Lemma 77 the following holds:

$$\{C \mid \mathrm{ord}(C) = (P, m - 1) \wedge C \text{ is a Horn clause over } \Sigma\} \tag{6.29}$$

     is finite up to subsumption.

□

6.4. Implementation

In order to obtain an actual decision procedure for BSH-Y2 that is efficient, sound, complete, and terminating for $\mathcal{C}^{\succ}_{\mathrm{Tr}}$ ontologies consisting of several million clauses, I have implemented the two-layered superposition reasoning calculus $\mathcal{C}^{\succ}_{\mathrm{Tr}}$ in SPASS–Y2.

The implementation of the calculus $\mathcal{C}^{\succ}_{\mathrm{Tr}}$ is based on the filtered context tree index that I have presented in Chapter 5 and that is also implemented in SPASS–Y2.

In order to further reduce the number of generated clauses, I have integrated empty sort and sort simplification directly into the implementation of hyperresolution and the chaining rules. This is based on the following observation: Any clause $\Theta, S_1, \ldots, S_n \parallel \Gamma \rightarrow \Delta$ from the BSH-Y2 with either (i) all S_i contain the variable x and $x \notin \mathrm{vars}(\Gamma \rightarrow \Delta)$ or (ii) all S_i are ground, can be reduced to either true or $\Theta \parallel \Gamma \rightarrow \Delta$ in polynomial time.

As a consequence, the integration of the sort reasoning into the implementation of the hyperresolution and chaining rules avoids the generation of clauses that can eventually be removed by empty sort, sort simplification, and static soft typing.

6.5. Summary

In this chapter, I have presented a superposition calculus for BSH-Y2 that is at the same time efficient, sound, complete, and terminating. Consequently, if implemented in the superposition reasoning framework, this calculus is a decision procedure for the BSH-Y2 fragment of first-order logic. The experiments of Chapter 8 confirm that this decision procedure is efficient also for BSH-Y2 ontologies consisting of several million clauses.

The key for successful reasoning about these ontologies is the two-layered reasoning calculus $\mathcal{C}^{\succ}_{\mathrm{Tr}}$ that performs a saturation with a dedicated calculus for each layer; the non-transitive reasoning layer and the transitive reasoning layer. Reasoning about sorts is performed independently by an adaption of the efficient sort reasoning calculus. In addition, in order to avoid the generation of clauses that eventually become redundant, the implementation integrates sort reasoning into the implementation of the hyperresolution rule and the chaining rules.

In the next chapter, I present a query answering procedure that answers complex queries with arbitrary quantifier alternations in terms of the minimal model of a BSH-Y2 ontology. In order to do this, the query answering procedure is defined with respect to the saturation N of the clause set with $\mathcal{C}^{\succ}_{\mathrm{Tr}}$. In the next chapter I proof that the saturation, with respect to $\mathcal{C}^{\succ}_{\mathrm{Tr}}$, of a BSH-Y2 ontology N is an efficient representation of the minimal model of N for query answering.

7. Query Answering in BSH-Y2 Ontologies

In this chapter, I introduce an efficient, sound, and complete query answering procedure that answers complex queries in BSH-Y2 ontologies with respect to minimal model semantics. The supported query language is a subset of first-order logic that can express complex formulas with arbitrary quantifier alternations. As the experiments in Chapter 8 confirm, the procedure answers complicated queries in the YAGO++ ontology, which is an extension of YAGO, within a few seconds.

In general, reasoning with minimal model semantics is above standard first-order reasoning. Therefore, I have developed a query answering procedure that is based upon a finite domain quantifier elimination algorithm. However, eliminating the quantifiers of a formula Φ by a complete ground instantiation generates a formula of size $\mathcal{O}(|\Sigma|^n)$ [SS03] where $|\Sigma|$ is the size of the signature Σ and n is the number of quantifiers in Φ. Because of the fact that the YAGO ontology has more than 2 million constants in the signature, a complete ground instantiation of Φ is practically not feasible.

The query answering procedure, I have developed, performs intermediate queries to the minimal model in order to restrict the number of ground instances of the query. These intermediate queries to the minimal model can be efficiently performed by exploiting the compact representation of the minimal model as a clause set that is saturated with respect to $\mathcal{C}_{\mathrm{Tr}}^{\succ}$.

In the first section of this chapter, I introduce the supported query language. After illustrating the operating principle of the query answering procedure, Section 7.2 presents the query answering calculus and the quantifier elimination algorithm. Section 7.3 proves the soundness and completeness of this algorithm. Section 7.4 presents details about the actual implementation of the algorithm in SPASS–Y2.

7.1. Query Language

The query language for BSH-Y2 ontologies is a subset of the first-order language and defined as follows:

Definition 79. *Let Σ be the signature of a BSH-Y2 ontology. The query language $\mathcal{L}_{\Sigma}^{Q}$ is defined as follows*

- $\Gamma \in \mathcal{L}_{\Sigma}^{Q}$ *where* Γ *is a multiset of ground atoms over* Σ
- $\forall x(\Gamma \rightarrow \Phi) \in \mathcal{L}_{\Sigma}^{Q}$ *if* Γ *is a multiset of atoms and* $\Phi \in \mathcal{L}_{\Sigma}^{Q}$

- $\exists x(\Gamma \wedge \Phi) \in \mathcal{L}_\Sigma^Q$ *if* Γ *is a multiset of atoms and* $\Phi \in \mathcal{L}_\Sigma^Q$

The example in Section 6.1 demonstrates that a transitive non-ground atom may cause the computation of the whole transitive closure. In order to avoid this behavior, additional restrictions on the query language are required.

Definition 80 (Shielded Variable)**.** *The variables of a formula* Φ *of the language* $\mathcal{L}_\Sigma^Q$ *are called* shielded *iff either* Φ *is ground or it is of the form* $\Phi = \exists x(\Gamma \wedge \Phi')$ *or* $\Phi = \forall x(\Gamma \rightarrow \Phi')$*, and all variables occurring under a transitive dependent predicate in* Γ *or occurring freely in* Φ'*, also occur under a non-transitive dependent predicate or a sort predicate in* Γ*.*

Definition 81 (Query)**.** *A formula* Φ *is a* query *if it is a sentence from the language* $\mathcal{L}_\Sigma^Q$ *such that all variables occurring in* Φ *are shielded.*

For simplicity, I assume that a variable is bound by at most one quantifier in a query.

Note, requiring shielded variables in a query is not a real restriction because it can always be achieved. All queries of Figure 1.2 are contained in this query language.

Furthermore, the query language of Definition 81 provides a way to express a restricted form of negation. For example, consider the query asking for the first German chancellor. In other words, this is a German chancellor who had no predecessor. Assume an atom $\bot$ such that $N_I \not\models \bot$. The following query, which belongs to the above query language, expresses this question:

$$\exists x(\text{GermanChancellor}(x) \wedge \forall y(\text{GermanChancellor}(y) \wedge \text{hasPredecessor}(x, y) \rightarrow \bot)).$$

For example, one could choose for $\bot$ the atom

$$\text{hasChild}(\text{AlbertEinstein}, \text{AlbertEinstein}) \tag{7.1}$$

because it holds that

$$N_I \not\models \text{hasChild}(\text{AlbertEinstein}, \text{AlbertEinstein}). \tag{7.2}$$

7.2. Query Answering Procedure

This section presents the query answering procedure that answers queries from $\mathcal{L}_\Sigma^Q$ in a BSH-Y2 ontology with respect to minimal model semantic. The first part of this section presents an example illustrating the operation principle of the query answering procedure. The second part introduces the query answering calculus. Finally, the last part presents the quantifier elimination procedure which is based on the query answering calculus.

7.2.1. Operating Principle of the Query Answering Procedure

This section shows the operating principle of the query answering procedure. It illustrates how an answer to a given query can be extracted from the minimal model of a BSH-Y2 ontology. Let N be the saturation, with respect to $\mathcal{C}^{\succ}_{\mathrm{Tr}}$, of a BSH-Y2 ontology, and $\square \notin N$. Further, let Σ be the signature of N and N_I be the minimal model of N (Definition 56). Assume the question: "Who was born in the same place as all his children?". This question can be expressed in first-order logic by the following query:

$$\Phi = \exists x, y(\mathrm{bornIn}(x, y) \ \wedge \ \forall z(\mathrm{hasChild}(x, z) \ \rightarrow \ \mathrm{bornIn}(z, y))). \tag{7.3}$$

As shown in Section 4.2.2, the standard first-order reasoning framework is not able to verify if $N_I \models \Phi$. In particular, it is not possible to change the signature. This is why Skolemization of the query is not an option.

Instead, a quantifier elimination procedure over finite domains [SS03], which performs a complete ground instantiation of the query, transforms the query into a quantifier free formula without changing the signature. This approach instantiates the quantified variables with all constants of the signature. The size of the resulting ground query is in $\mathcal{O}(|\Sigma|^n)$, where $|\Sigma|$ is the size of the signature Σ and n the number of quantifiers in Φ. This means that the size of the ground instantiated query is single exponential in the number of quantifiers. However, an approach by a complete ground instantiation, is not feasible in order to answer queries in an ontology like YAGO because the signature of YAGO contains more than two million constants. Therefore, the approach I have developed, restricts the number of considered ground instances, while remaining complete, by performing intermediate queries to the minimal model N_I.

In order to eliminate the outermost existential quantifier in the current example, it is actually not necessary to instantiate the quantified variables x and y with all constants of Σ. Instead, one needs to consider only those constants for the variables x and y such that the respective instance of the atom $\mathrm{bornIn}(x, y)$ is entailed by the minimal model. Formally, this is the following set of grounding substitutions:

$$\{\sigma \mid \mathrm{dom}(\sigma) = \{x, y\}, N_I \models \mathrm{bornIn}(x\sigma, y\sigma)\}. \tag{7.4}$$

The original query Φ can be transformed into the following disjunction which is entailed by N_I if and only if $N_I \models \Phi$:

$$\bigvee_{N_I \models \mathrm{bornIn}(x\sigma, y\sigma)} \forall y(\mathrm{hasChild}(x\sigma, z) \ \rightarrow \ \mathrm{bornIn}(z, y\sigma)), \tag{7.5}$$

where all σ are substitutions such that $\mathrm{bornIn}(x, y)\sigma$ is ground.

Likewise, the universal quantifier can be eliminated by a finite instantiation as follows:

$$\bigvee_{N_I \models \mathrm{bornIn}(x\sigma, y\sigma)} \ \bigwedge_{N_I \models \mathrm{hasChild}(x\sigma, z\sigma')} \mathrm{bornIn}(z\sigma', y\sigma). \tag{7.6}$$

Verifying if the ground formula 7.6 is entailed by the minimal model N_I can be performed by several queries of the following form:

$$N_I \models \text{bornIn}(z\sigma', y\sigma). \tag{7.7}$$

The standard first-order reasoning framework can decide this reasoning problem (Proposition 58).

The following lemma proves that the respective ground instantiations (in the example σ and σ') can be efficiently retrieved from N for transitive independent predicates.

Lemma 82. *Let N be the saturation, with respect to $\mathcal{C}^{\succ}_{\text{Tr}}$, of a BSH-Y2 clause set, $\square \notin N$,* sel *be the selection function (Definition 66), A be a transitive independent ground atom, and N_I the minimal model of N. Then $N_I \models A$ iff there is a ground substitution σ and a clause $\Theta \parallel \rightarrow B \in N$ with $B\sigma = A$ and $\text{S}_N \models \Theta\sigma$.*

Proof. " $\Leftarrow$ " N_I defines a Herbrand model with $N_I \models N$ by Theorem 75.

" $\Rightarrow$ " Assume $N_I \models B\sigma$. Then there is a ground instance $C\sigma = (\Theta \parallel \Gamma \rightarrow B)\sigma$ of a clause $C \in N$ that produces $B\sigma$. This means that $B\sigma$ is maximal in $C\sigma$, $(\Theta \cup \Gamma)\sigma \subseteq R^*_{C\sigma}$, and $B\sigma \notin R^*_{C\sigma}$. This implies $N_I \models \Theta\sigma$, and, consequently, $\text{S}_N \models \Theta\sigma$ by Lemma 69.

Now, we show by contradiction that $\Gamma\sigma$ in $C\sigma$ must be empty. So, assume $\Gamma\sigma \neq \emptyset$. By assumption $B\sigma$ is transitive independent. Consequently, all $B'\sigma \in \Gamma\sigma$ are non-transitive. By the selection function sel one $B'\sigma \in \Gamma\sigma$ is selected. This contradicts the fact that $C\sigma$ is productive by definition of N_I (Definition 56). □

Note, the clause $\Theta \parallel \rightarrow B$ together with the respective unifier σ can be efficiently retrieved from N using the filtered context tree index procedures of Chapter 5.

7.2.2. Query Answering Calculus

Let N be the saturation, with respect to $\mathcal{C}^{\succ}_{\text{Tr}}$, of a set of clauses from BSH-Y2, and $\square \notin N$. From Theorem 75 it follows that N is a sufficient representation of its minimal model, i.e.,

$$N_I \models N \cup \mathcal{A}_{\text{Tr}} \cup \text{N}_{\text{una}}.$$

Further, let S_N be the static sort theory of N. It holds that $\text{S}_N \subseteq N$ as shown in Section 4.1.3.

The query answering calculus is composed of a deterministic rule system with respect to N and S_N. It consists of three calculus rules; one for each type of query: existential query, universal query and ground query.

Basically, the calculus derives for each query $\forall x(\Gamma \rightarrow \Phi)$ and $\exists x(\Gamma \wedge \Phi)$ the instances $\Phi\sigma$ of the subquery Φ with $N_I \models \Gamma\sigma$. The calculus obtains the instances σ by performing reasoning tasks on N.

For all rules, we assume that A_i are transitive independent atoms, T_i are transitive dependent atoms, and S_i are sort atoms. Note, each subquery $\Phi'\sigma$, derived from the query answering calculus, is again a query because all variables of Φ are shielded by Definition 81. Likewise, for all transitive dependent atoms T_i of a query Φ and a substitution σ, it holds that $T_i\sigma$ is ground if σ is grounding for all transitive independent atoms and all sort atoms of Φ.

Verifying the side-conditions of the query answering calculus rules requires to perform entailment operations. The sort entailment of condition 1 and condition 3 is a well-sortedness check which is quasi-linear [SS89]. The entailment check in condition 4 is performed by exhaustively applying the saturation calculus $\mathcal{C}^{\succ}_{\mathrm{Tr}}$ with a set-of-support strategy. Note, from Proposition 58 we know that $\mathcal{C}^{\succ}_{\mathrm{Tr}}$ together with the superposition reasoning framework is a decision procedure for this minimal model reasoning problem. In addition, the set-of-support strategy is complete in this case because N is saturated.

Existential Query

$$\frac{\Phi = \exists \overline{x}(\, S_1 \wedge \cdots \wedge S_{n_1} \wedge A_1 \wedge \cdots \wedge A_{n_2} \wedge T_1, \ldots, T_{n_3} \wedge \Phi') \quad \Theta_i \parallel \rightarrow A'_i}{\Phi'\sigma}$$

if $1 \leq i \leq n_2$ and there is a grounding substitution σ such that

1. $\mathrm{S}_N \models S_i\sigma$ for all $i \in \{1, \ldots, n_1\}$
2. $A_i\sigma = A'_i\sigma$ for all $i \in \{1, \ldots, n_2\}$
3. $\mathrm{S}_N \models \Theta_i\sigma$ for all $i \in \{1, \ldots, n_2\}$
4. $N_I \models T_i\sigma$ for all $i \in \{1, \ldots, n_3\}$

Universal Query

$$\frac{\Phi = \forall \overline{x}(\, S_1 \wedge \cdots \wedge S_{n_1} \wedge A_1 \wedge \cdots \wedge A_{n_2} \wedge T_1 \wedge \cdots \wedge T_{n_3} \rightarrow \Phi') \quad \Theta_i \parallel \rightarrow A'_i}{\Phi'\sigma}$$

if $1 \leq i \leq n_2$ and there is a grounding substitution σ such that

1. $\mathrm{S}_N \models S_i\sigma$ for all $i \in \{1, \ldots, n_1\}$
2. $A_i\sigma = A'_i\sigma$ for all $i \in \{1, \ldots, n_2\}$
3. $\mathrm{S}_N \models \Theta_i\sigma$ for all $i \in \{1, \ldots, n_2\}$
4. $N_I \models T_i\sigma$ for all $i \in \{1, \ldots, n_3\}$

Ground Query

$$\frac{\Phi = S_1 \wedge \cdots \wedge S_{n_1} \wedge A_1 \wedge \cdots \wedge A_{n_2} \wedge T_1 \wedge \cdots \wedge T_{n_3} \quad \Theta_i \parallel \rightarrow A'_i}{\text{true}}$$

if $1 \leq i \leq n_2$ and there is a grounding substitution σ such that

1. $\mathrm{S}_N \models S_i$ for all $i \in \{1, \ldots, n_1\}$
2. $A_i = A'_i\sigma$ for all $i \in \{1, \ldots, n_2\}$
3. $\mathrm{S}_N \models \Theta_i$ for all $i \in \{1, \ldots, n_2\}$
4. $N_I \models T_i$ for all $i \in \{1, \ldots, n_3\}$

7.2.3. Query Answering Algorithm

Algorithm 15 implements the query answering procedure that answers a query Φ with respect to minimal model semantics by performing a finite quantifier elimination algorithm that is based on the query answering calculus. Let N be the saturation, with respect to $\mathcal{C}^{\succ}_{\mathrm{Tr}}$, of a BSH-Y2 ontology, and $\square \notin N$. In this case, the algorithm is sound and complete.

The algorithm expects as its input a query Φ and the saturated clause set N. First, the algorithm checks whether the given query Φ is an existential quantified, a universal quantified, or a ground query. Then it computes the set of all subqueries obtained by applying the above defined query answering calculus rules.

The set $\mathrm{ext}(\Phi, N)$ is the set of all subqueries from applying the rule *Existential query* to Φ and N. Likewise, the set $\mathrm{unv}(\Phi, N)$ is the set of all subqueries from applying the rule *Universal query* to Φ and N. Finally, $\mathrm{gnd}(\Phi, N)$ is the result of the application of *Ground query*. If $\mathrm{gnd}(\Phi, N)$ is true then the algorithm returns true otherwise it returns false.

Because of the fact that the implication in a universal query is not symmetric the algorithm processes a query from the outer query to the inner subquery. Verifying if $N_I \models \forall x(\Gamma \rightarrow \Phi')$ requires to check whether each ground instance of Γ is also contained in the set of instances of Φ'. This requires that all ground instances of Γ have to be computed. Therefore, computing the instances of Φ' at first does not help to improve performance.

Algorithm 15: AnswerQuery

Input: Query Φ, saturated clause set N

```
1  if Φ = ⊤ then
2  |  return true
3  else if Φ = ∃x̄.Γ ∧ Φ′ then
4  |  foreach Φ′σ ∈ ext(Φ, N) do
5  |  |  if AnswerQuery(Φ′σ,N) then
6  |  |  |  return true;
7  |  |  end
8  |  end
9  |  return false;
10 else if Φ = ∀x̄.Γ → Φ′ then
11 |  foreach Φ′σ ∈ unv(Φ, N) do
12 |  |  if ¬AnswerQuery(Φ′σ, N) then
13 |  |  |  return false;
14 |  |  end
15 |  end
16 |  return true;
17 else if gnd(Φ, N) = true then
18 |  return true
19 else
20 |  return false
21
```

7.3. Soundness and Completeness

Theorem 83 (Soundness of AnswerQuery)**.** *Let Φ be a query and N be a clause set that is saturated with respect to $\mathcal{C}^{\succ}_{\mathrm{Tr}}$ from BSH-Y2. Then the following holds:*

$$\mathrm{AnswerQuery}(\Phi, N) = \mathrm{true} \Rightarrow N_I \models \Phi.$$

Proof. Assume $\mathrm{AnswerQuery}(\Phi, N) = \mathrm{true}$. For a query Φ, I show that $N_I \models \Phi$ by induction on the structure of Φ. As base case, assume Φ is a ground query with

$$\Phi = S_1 \wedge \cdots \wedge S_{n_1} \wedge A_1 \wedge \cdots \wedge A_{n_2} \wedge T_1 \wedge \cdots \wedge T_{n_3}$$

From $\mathrm{AnswerQuery}(\Phi, N) = \mathrm{true}$ it follows $\mathrm{gnd}(\Phi, N) = \mathrm{true}$. Consequently, the side conditions 1 – 4 of the rule *Ground query* hold. It remains to show that $N_I \models \Phi$ is a consequence of these conditions. In order to show this, I prove that $N_I \models A$ for all atoms A of Φ.

- Let A be a sort atom in Φ. $\mathrm{S}_N \models A$ by condition 1 and consequently, $N_I \models A$.
- Let A be a transitive independent atom that is no sort atom. By conditions 2 and condition 3 together with Lemma 82 it follows: $N_I \models A$.
- Let A be a transitive atom. From condition 4 it follows: $N_I \models A$.

Now, assume $\Phi = \exists \overline{x}(\Gamma \wedge \Phi')$ is an existential query with

$$\Gamma = S_1 \wedge \cdots \wedge S_{n_1} \wedge A_1 \wedge \cdots \wedge A_{n_2} \wedge T_1 \wedge \cdots \wedge T_{n_3}$$

In order to show $N_I \models \Phi$ one can equivalently show that there is a substitution σ with $N_I \models \Gamma\sigma$ and $N_I \models \Phi'\sigma$. From the assumption $\mathrm{AnswerQuery}(\Phi, N) = \mathrm{true}$ it follows that $\Phi'\sigma \in \mathrm{ext}(\Phi, N)$. Consequently, by definition of ext, the conditions $1 - 4$ of *Existential query* are fulfilled for σ and, consequently, $N_I \models \Gamma\sigma$. By the induction hypothesis it follows that $N_I \models \Phi'\sigma$.

If $\Phi = \forall \overline{x}.\Gamma \rightarrow \Phi'$ is a universal query then we need to show that for all σ with $N_I \models \Gamma\sigma$ also $N_I \models \Phi'\sigma$. From the assumption $\mathrm{AnswerQuery}(\Phi, N) = \mathrm{true}$ it follows for $\Phi'\sigma \in \mathrm{unv}(\Phi, N)$ that condition 1–4 of *Universal Query* hold for $\Gamma\sigma$ and, as a consequence, $N_I \models \Gamma\sigma$. By definition of unv the set $\mathrm{unv}(\Phi, N)$ contains all $\Phi'\sigma$ with $N_I \models \Gamma\sigma$. For each $\Phi'\sigma$ it holds that $N_I \models \Phi'\sigma$ by the induction hypothesis. □

Theorem 84 (Completeness of AnswerQuery)**.** *Let Φ be a query, N the saturation, with respect to $\mathcal{C}^{\succ}_{\mathrm{Tr}}$, of a clause set from BSH-Y2. Let N_I be the minimal model of N then*

$$N_I \models \Phi \Rightarrow \mathrm{AnswerQuery}(\Phi, N) = \mathrm{true}$$

Proof. The proof of the theorem is by structural induction on Φ. Let Φ be a ground query with $N_I \models \Phi$ then Φ has the following form

$$\Phi = S_1 \wedge \cdots \wedge S_{n_1} \wedge A_1 \wedge \cdots \wedge A_{n_2} \wedge T_1 \wedge \cdots \wedge T_{n_3}$$

This means that $N_I \models S_i$ for $i \in \{1, \ldots, n_1\}$ and $N_I \models A_i$ for $i \in \{1, \ldots, n_2\}$ and $N_I \models T_i$ for $i \in \{1, \ldots, n_3\}$. By Lemma 82 it follows that $\Theta_i \parallel \rightarrow A'_i \in N$ and σ with $A'_i\sigma = A_i$ and $\mathrm{S}_N \models \Theta_i\sigma$ for all $i \in \{1, \ldots, n_2\}$. Because for the static sort theory S_N of N it holds that $\mathrm{S}_N \subseteq N$, it follows form $N_I \models S_i$ that $\mathrm{S}_N \models S_i$. By definition of gnd it follows:

$$\mathrm{gnd}(\Phi, N) = \mathrm{true}. \tag{7.8}$$

If Φ is an existential query then it has the following form

$$\Phi = \exists \overline{x}(\Gamma \wedge \Phi')$$

Because of the fact that $N_I \models \Phi$, it follows that there is a grounding substitution σ of $\Gamma \wedge \Phi$ with $N_I \models \Gamma\sigma \wedge \Phi'\sigma$. This means that $N_I \models \Gamma\sigma$ and $N_I \models \Phi'\sigma$. The set Γ has the following form

$$\Gamma = S_1 \wedge \cdots \wedge S_{n_1} \wedge A_1 \wedge \cdots \wedge A_{n_2} \wedge T_n \wedge \cdots \wedge T_{n_3} \tag{7.9}$$

where S_i are atoms of sort predicates, A_i are atoms of binary transitive independent predicates and T_i are atoms of binary transitive dependent predicates. As a consequence we get $N_I \models S_i\sigma$ for $i \in \{1, \dots n_1\}$, $N_I \models A_i\sigma$ for $i \in \{1, \dots n_2\}$ and $N_I \models T_i\sigma$ for $i \in \{1, \dots n_3\}$. From $N_I \models A_i\sigma$ and Lemma 82 we get $\Theta_i \parallel \rightarrow A_i' \in N$ and $A_i\sigma = A_i'\sigma$ with $\mathrm{S}_N \models \Theta_i\sigma$. From $N_I \models S_i\sigma$ and $N_I \models N$ we get $\mathrm{S}_N \models S_i\sigma$ and also because of $N_I \models N$ we have that $N \models T_i\sigma$. As a consequence, an application of the rule *existential query* derives $\Phi'\sigma$. From $N_I \models \Phi'\sigma$ and the fact that all variables of Φ' are shielded we get that $\Phi'\sigma$ is a query and, by induction, it follows that $\mathrm{AnswerQuery}(\Phi', N) = \mathrm{true}$.

If Φ is an universal query then it has the following form

$$\Phi = \forall \overline{x}(\Gamma \rightarrow \Phi')$$

Because of the fact that $N_I \models \Phi$ for all $\sigma = \{x_i \mapsto c_i \mid c_i \in \Sigma, i \in \{1, \dots, n\}\}$ we get that $N_I \not\models \Gamma$ or $N_I \models \Phi'$. Assume that $N_I \models \Gamma$ because otherwise we are done. In this case the theorem follows analogously to the case of existential queries. □

7.4. Implementation in Spass–Y2

The implementation of the query answering procedure for BSH-Y2 ontologies follows exactly Algorithm 15. The rules are implemented following the implementation of hyperresolution style rules. Furthermore, instead of generating new subqueries for each instance, the implementation represents an instance of a subquery implicitly by the respective substitution. Composing two substitutions is linear in the size of the substitutions, i.e., the number of bound variables.

One exception to the straight forward implementation is the condition 4 that checks whether a transitive dependent atom T is entailed by N_I. The implementation does not initiate the whole reasoning engine of Spass–Y2 for this purpose. Instead, it uses a special procedure that simulates several derivation steps in hyperresolution style macro steps while efficiently representing the newly derived clauses by their respective substitutions together with the original clause.

By Lemma 82, the retrieval of instances can be done by searching the saturated clause set N for respective instances. The filtered context tree index, introduced in Chapter 5, efficiently performs these tasks. Additionally, the index maintains the number of ground instances of a predicate. This allows the procedure to determine an optimized order for processing the atoms A_i of a given query. More precisely, it orders the atoms A_i by the number of instances in increasing order and begins with the atom that has the least instances. This avoids the unnecessary inspection of non successful instances.

Actually, in case of an existential query $\exists x\ \Phi$, the implementation of AnswerQuery returns all ground instances σ with $N_I \models \Phi\sigma$. In the case of an universal query $\forall x\ \Phi$ that is not entailed by N_I the implementation of AnswerQuery returns a counter example σ with $N_I \not\models \Phi\sigma$.

7.5. Summary

This chapter has introduced an efficient query answering procedure for complex queries that contain arbitrary quantifier alternations. The query answering procedure answers these queries in a clause set from BSH-Y2 with respect to minimal model semantics. If N is saturated in terms of $\mathcal{C}^{\succ}_{\mathrm{Tr}}$, and $\square \notin N$ then the query answering procedure is sound and complete.

8. Evaluation

In this chapter, I present the results obtained from evaluating SPASS–Y2 that is based on the automated theorem prover SPASS [WDF+09]. SPASS–Y2 implements the procedures that I have presented in this thesis: the saturation procedure (Chapter 6), the query answering procedure (Chapter 7), and the underlying filtered context tree index (Chapter 5). The experiments show that SPASS–Y2 efficiently decides the satisfiability of large ontologies and answers complex queries that contain arbitrary quantifier alternations in minimal model semantics.

The chapter is structured as follows. First, I present the the sample ontologies that I have used for the evaluation of SPASS–Y2. These sample ontologies are based on the following three ontologies: YAGO, SUMO, and CYC. In Section 8.2, I present the results obtained from saturating the above-mentioned ontologies with SPASS–Y2 and from comparing these results with other reasoning tools. Finally, in Section 8.3, I evaluate the query answering capabilities of SPASS–Y2 with respect to both standard first-order semantics and minimal model semantics.

The experiments presented in this chapter are computed on a 2 x Intel Xeon Processor X5660 (12 MB Cache, 2.80 GHz) Debian Linux machine with 96 GB RAM. SPASS–Y2 requires a 64 bit architecture for the experiments because it addresses around 20 GB RAM for the saturation of the YAGO++ ontology.

8.1. The Sample Ontologies

This section describes the ontologies I have used for the experiments presented in this chapter. These are the YAGO++ ontology and the BSH-Y2 subsets of SUMO and CYC.

8.1.1. The YAGO++ Ontology

The BSH-Y2 representation of YAGO, as shown in Chapter 3, contains clauses of the following types: ground facts, subsort relations, functionality, and transitivity axioms. In addition, this representation assumes a unique name assumption.

In order to evaluate also the other constructs of BSH-Y2, namely constraints and defined relations, in the context of YAGO, I have created the YAGO++ ontology. Figure 8.1 depicts the defined relation that YAGO++ contains in addition to the BSH-Y2

$$\text{male}(x), \text{female}(y)\, \text{isMarriedTo}(x, y) \rightarrow \text{wifeOf}(y, x)$$
$$\text{male}(x), \text{female}(y)\, \text{isMarriedTo}(x, y) \rightarrow \text{husbandOf}(y, x)$$

$$\text{male}(x), \text{hasChild}(x, y) \rightarrow \text{fatherOf}(y, x)$$
$$\text{male}(y), \text{hasChild}(x, y) \rightarrow \text{sonOf}(x, y)$$
$$\text{female}(x), \text{hasChild}(x, y) \rightarrow \text{motherOf}(y, x)$$
$$\text{female}(y), \text{hasChild}(x, y) \rightarrow \text{daughterOf}(x, y)$$

$$\text{fatherOf}(x, y) \rightarrow \text{parentOf}(x, y)$$
$$\text{motherOf}(x, y) \rightarrow \text{parentOf}(x, y)$$

$$\text{produced}(x, y), \text{movie}(y) \rightarrow \text{producedMovie}(x, y)$$

$$\text{bornIn}(x, y), \text{locatedIn}(y, z), \text{country(z)} \rightarrow \text{bornInCountry}(x, z)$$

Figure 8.1.: Defined relations in YAGO++

representation of YAGO. The additional constraints of YAGO++ are negative facts, asymmetry and irreflexivity axioms.

As shown in Chapter 4, one can formulate asymmetry and irreflexivity constraints for transitive relations without causing the saturation procedure to generate the transitive closure. This can be achieved by defining an extra predicate representing the transitive closure. For example, the YAGO++ ontology contains the following axioms instead of the transitivity axiom for locatedIn. The predicate locatedInTC represents together with the respective transitivity axiom the transitive closure of locatedIn:

$$\text{locatedIn}(x, y) \rightarrow \text{locatedInTC}(x, y)$$
$$\text{locatedInTC}(x, y), \text{locatedInTC}(y, z) \rightarrow \text{locatedInTC}(x, z).$$

Now, the asymmetry and irreflexivity constraint can be used for the non-transitive predicate locatedIn as follows:

$$\text{locatedIn}(x, x) \rightarrow$$
$$\text{locatedIn}(x, y), \text{locatedIn}(y, x) \rightarrow .$$

The YAGO++ ontology contains in sum $9,918,724$ clauses. Figure 8.2 shows how many clauses from each type YAGO++ contains. The YAGO++ ontology will be integrated into the next releases of the TPTP that is the benchmark library for automated theorem proving tools.

Clause	Quantity
$\rightarrow P(a,b)$	5,163,706
$\rightarrow S(a)$	4,505,488
$S(x) \rightarrow T(y)$	249,431
$P(x,y), P(x,z) \rightarrow y \approx z$	60
$P_1(t_{11},t_{12}) \ldots, P_k(t_{k1},t_{k2}) \rightarrow$	20
$P_1(t_{11},t_{12}) \ldots, P_k(t_{k1},t_{k2}) \rightarrow P(s_1,s_2)$	10
$P(x,y), P(y,z) \rightarrow P(x,z)$	3
unique name assumption	

Figure 8.2.: The type and number of clauses contained in YAGO++

8.1.2. The SUMO Ontology

The benchmark library for automated theorem proving TPTP contains a representation of SUMO [NP01a] as a set of first-order clauses [PS07]. This representation of SUMO is not completely contained in BSH-Y2. For this reason, I only considered the subset that is contained in BSH-Y2 for the experiments. This set is called SUMO-Y2 and contains about 90% (83k clauses) of SUMO from the TPTP.

Furthermore, the type information of individuals in SUMO is encoded with the binary relation s__instance. As presented in Chapter 6, reasoning about type information is much more efficient if the respective type information is represented with sorts. For this reason, I have transformed the type information of SUMO-Y2 into sorts. Additionally, I have removed all non-static sort clauses from SUMO-Y2.

8.1.3. The CYC Ontology

There is also a first-order translation [RRG05] of CYC [RRG05] contained in the TPTP library. The CYC ontology contains several microtheories. Each microtheory encodes knowledge of a particular domain. In addition, there is also a base theory containing basic knowledge which is common to all microtheories. For the experiments in this Chapter, I consider the BSH-Y2 subset of the base theory of CYC. This subset is called CYC-Y2 and it contains about 30% (1 million clauses) of the overall CYC ontology as contained in the TPTP. The type information of CYC is already represented as sorts.

8.2. Saturation

In the experiments, I have compared clasp 2.0.4 with the grounder gringo [GKK+11], DLV [LPF+06], Vampire 0.6 [RV01], E 1.4 [Sch02], iProver 0.8.1 [Kor08], Spass 3.8 [WDF+09], and Spass-Y2. Spass 3.8 contains already the filtered context tree index

	YAGO++			SUMO-Y2			CYC-Y2		
Tool	Derived	Result	Time	Derived	Result	Time	Derived	Result	Time
clasp	1,118,858,572	kbs	70 min	1,322,070	sat	20 sec		unsat	1 min
DLV		t.o.	100 min		sat	30 sec		t.o.	100 min
Vampire		kbs	12 sec		kbs	1 min		kbs	4 min
E		kbs	6 min		t.o.	100 min		kbs	3 min
iProver		kbs	1 min	967,678	t.o.	100 min		kbs	8 sec
SPASS 3.8	49,848,842	sat	60 min	1,530,025	t.o.	100 min	18,907,803	t.o.	100 min
SPASS-Y2	2,724,048	sat	16 min	790,691	sat	53 min	328,904	unsat	1 min

Figure 8.3.: Evaluation of SPASS-Y2

as presented in Chapter 5. But it does not have an implementation of the new calculus presented in Chapter 6.

All provers were called with the recommended default settings and a time limit of 100 min. Each of these tools was run with each of the ontologies YAGO++, SUMO-Y2, and CYC-Y2.

SPASS-Y2 has found inconsistencies in YAGO++ and SUMO-Y2. Figure 8.4 depicts a proof of an inconsistency that SPASS–Y2 has found in SUMO-Y2. I have manually removed these inconsistencies from YAGO++ and SUMO-Y2. After that, SPASS–Y2 could saturate both ontologies YAGO++ and SUMO-Y2. SPASS-Y2 has identified 35 inconsistencies in CYC which I have also manually removed. It still contains inconsistencies, and as a consequence, SPASS-Y2 does not saturate CYC-Y2 because it finds an inconsistency.

The results of the experiments are depicted in Figure 8.3. The first column shows the tool and the third column the results for the respective ontology. The column derived shows the number of newly generated formulas during problem processing. This column contains empty entries because this information was not always available when the prover timed out (t.o.) after 100 min or was killed by operating system/self killed (kbs), which is marked in the result column.

In summary, SPASS-Y2 can effectively decide the satisfiability for all three ontologies, where all other systems fail on at least one input set. Clasp performed nicely on SUMO-Y2 and CYC-Y2, but failed on YAGO++ because, due to the 2m constants and transitive relations, gringo was unable to completely ground instantiate YAGO++.

```
2138[0:Inp] ||  -> Ss__City(s__JerusalemIsrael)*.
4198[0:Inp] Ss__LandArea(U) ||  -> Ss__GeographicArea(U)*.
6560[0:Inp] Ss__GeographicArea(U) ||  -> Ss__Region(U)*.
7206[0:Inp] ||  -> s__geographicSubregion(s__JerusalemIsrael,s__Israel)*.
7760[0:Inp] ||  -> Ss__GeographicArea(s__WestBank)*.
12964[0:Inp] Ss__Region(U) ||  -> Ss__Object(U)*.
14914[0:Inp] ||  -> s__geographicSubregion(s__JerusalemIsrael,s__WestBank)*.
19333[0:Inp] Ss__Agent(U) ||  -> Ss__Object(U)*.
19642[0:Inp] ||  -> Ss__Nation(s__WestBank)*.
25040[0:Inp] ||  -> Ss__LandArea(s__WestBank)*.
27820[0:Inp] Ss__City(U) ||  -> Ss__LandArea(U)*.
28157[0:Inp] ||  -> s__meetsSpatially(s__WestBank,s__Israel)*.
31172[0:Inp] Ss__Nation(U) ||  -> Ss__GeopoliticalArea(U)*.
33618[0:Inp] ||  -> Ss__Nation(s__Israel)*.
44236[0:Inp] ||  -> Ss__GeopoliticalArea(s__WestBank)*.
49814[0:Inp] Ss__GeopoliticalArea(U) ||  -> Ss__GeographicArea(U)*.
50380[0:Inp] Ss__GeopoliticalArea(U) ||  -> Ss__Agent(U)*.
80769[0:Inp] Ss__Object(U) Ss__Object(V) ||
 s__properPart(V,U)+ -> s__part(V,U)*.
80874[0:Inp] Ss__Object(U) Ss__Object(V) ||
s__meetsSpatially(V,U)*+ -> s__meetsSpatially(U,V)*.
80993[0:Inp] Ss__GeographicArea(U) Ss__GeographicArea(V) ||
 s__geographicSubregion(V,U)*+ -> s__properPart(V,U).
81215[0:Inp] Ss__Object(U) Ss__Object(V) ||
 s__meetsSpatially(U,V)*+ s__overlapsSpatially(U,V) -> .
81653[0:Inp] Ss__Object(U) Ss__Object(V) Ss__Object(W) ||
s__part(W,U)*+ s__part(W,V)* -> s__overlapsSpatially(V,U)*.
82911[0:SSHy:80993.2,14914.0,27820.0,4198.0,2138.0,7760.0,19642.0,25040.0,44236.0]
   ||  -> s__properPart(s__JerusalemIsrael,s__WestBank)*.
82978[0:SSHy:80993.2,7206.0,27820.0,4198.0,2138.0,31172.0,49814.0,33618.0]
   ||  -> s__properPart(s__JerusalemIsrael,s__Israel)*.
159004[0:SSHy:80769.2,82911.0,27820.0,4198.0,6560.0,12964.0,2138.0,50380.0,
              19333.0,7760.0,19642.0,25040.0,44236.0]
   ||  -> s__part(s__JerusalemIsrael,s__WestBank)*.
159071[0:SSHy:80769.2,82978.0,27820.0,4198.0,6560.0,12964.0,2138.0,31172.0,
             49814.0,6560.0,12964.0,33618.0]
   ||  -> s__part(s__JerusalemIsrael,s__Israel)*.
160060[0:SSHy:80874.2,28157.0,50380.0,19333.0,7760.0,19642.0,25040.0,44236.0,
             31172.0,49814.0,6560.0,12964.0,33618.0]
   ||  -> s__meetsSpatially(s__Israel,s__WestBank)*.
629035[0:SSHy:81653.3,159004.0,81653.4,159071.0,27820.0,4198.0,6560.0,12964.0,
             2138.0,31172.0,49814.0,6560.0,12964.0,33618.0,50380.0,19333.0,
             7760.0,19642.0,25040.0,44236.0]
   ||  -> s__overlapsSpatially(s__Israel,s__WestBank)*.
657017[0:SSHy:81215.2,160060.0,81215.3,629035.0,50380.0,19333.0,7760.0,19642.0,
             25040.0,44236.0,31172.0,49814.0,6560.0,12964.0,33618.0]
   ||  -> .
```

Figure 8.4.: Proof of an inconsistency that SPASS–Y2 found in SUMO-Y2

8.3. Query Answering

In this section, I present the results of testing the query answering abilities of SPASS–Y2 with respect to both standard first-order semantics and minimal model semantics.

8.3.1. Standard Semantics

For the evaluation in terms of the standard first-order semantics, I tested the 20 queries of the SUMO category of the last CASC competition. First, I saturated the consistent SUMO-Y2 ontology with SPASS–Y2. Then I have added the respective query to this saturated clause set as conjecture. Finally, I applied the saturation procedure of SPASS-Y2, as presented in Chapter 6 with a, in this case, complete set-of-support strategy. This approach terminates on 13 problems with a proof and on further five with a consistent saturated set. The latter result is due to the fact that SUMO-Y2 does not contain all SUMO clauses. All results were obtained within one second. The remaining two problems cannot be formulated in BSH-Y2.

8.3.2. Minimal model semantics

Example Run

The following depicts an example execution of the query answering procedure answering a query in YAGO++ with minimal model semantics. Let the clause set N be the saturation of YAGO++ with $\mathcal{C}^{\succ}_{\mathrm{Tr}}$ (Chapter 6). Assume the following query:

$$\exists x, y(\mathrm{bornIn}(x, y) \wedge \forall z(\mathrm{hasChild}(x, z) \rightarrow \mathrm{bornIn}(z, y))).$$

The predicate bornIn is transitive independent in N. An application of the rule existential query derives a set of subqueries obtained by clauses of the form $\Theta \parallel \rightarrow \mathrm{bornIn}(a, b)$ from N such that $S_N \models \Theta$. Actually, $\Theta = \emptyset$ because bornIn is not defined in N. Consequently, the rule existential query removes the outermost existential quantifier by instantiating x and y. The resulting subqueries look as follows:

$$\forall z(\mathrm{hasChild}(\mathrm{GalileoGalilei}, z) \rightarrow \mathrm{bornIn}(\mathrm{z}, \mathrm{Pisa})) \tag{8.1}$$
$$\forall z(\mathrm{hasChild}(\mathrm{AlbertEinstein}, z) \rightarrow \mathrm{bornIn}(\mathrm{z}, \mathrm{Ulm})) \tag{8.2}$$
$$\forall z(\mathrm{hasChild}(\mathrm{PierreCurie}, z) \rightarrow \mathrm{bornIn}(\mathrm{z}, \mathrm{Paris})) \tag{8.3}$$
$$\forall z(\mathrm{hasChild}(\mathrm{MaxPlanck}, z) \rightarrow \mathrm{bornIn}(\mathrm{z}, \mathrm{Kiel})) \tag{8.4}$$
$$\forall z(\mathrm{hasChild}(\mathrm{JamesClerkMaxwell}, z) \rightarrow \mathrm{bornIn}(\mathrm{z}, \mathrm{Edinburgh})) \tag{8.5}$$

$$\vdots$$

If one of these subqueries holds, then also the whole query holds. So, assume subquery 8.3 to continue with. The rule universal query searches N for all ground instances

of hasChild(PierreCurie, z). The predicate hasChild is also not defined. So, universal query searches all ground instances of the clause $\| \rightarrow$ hasChild(PierreCurie, z). In N it finds the two clauses

$$\| \rightarrow \text{hasChild}(\text{PierreCurie}, \text{IrèneJoliot-Curie}))$$
$$\| \rightarrow \text{hasChild}(\text{PierreCurie}, \text{ÈveCurie}))$$

The resulting subqueries are

$$\text{bornIn}(\text{IrèneJoliot-Curie}, \text{Paris})) \tag{8.6}$$
$$\text{bornIn}(\text{ÈveCurie}, \text{Paris})) \tag{8.7}$$

Due to the universal quantifier both of these queries have to hold in the minimal model N_I. The rule ground query searches N and finds the following two clauses:

$$\| \rightarrow \text{bornIn}(\text{IrèneJoliot-Curie}, \text{Paris})$$
$$\| \rightarrow \text{bornIn}(\text{ÈveCurie}, \text{Paris}).$$

The rule ground query derives true for both subquery 8.6 and subquery 8.7. As a consequence, the query holds in N with minimal model semantics. In other words, it is entailed by the minimal model N_I.

Experimental Results

This section shows the experimental results obtained by applying the query answering procedure that I have presented in Chapter 7 to sample queries. Each query regards a particular feature of the query language or of BSH-Y2. This includes quantifier alternations, transitive dependent defined relations, transitive independent defined relations, and the restricted negation. The answering times of the query answering procedure for each of the following presented queries is depicted in Figure 8.5. For these experiments, the query answering procedure works in terms of the saturation, with respect to $\mathcal{C}_{\text{Tr}}^{\succ}$, of the YAGO++ ontology. The saturated ontology has been loaded into the index data structures of SPASS–Y2 prior to answering the queries. So, the given runtimes are just for question answering. The first column of Figure 8.5 specifies the query, the second column shows the query answering times in the format seconds.milliseconds. The current implementation of SPASS–Y2 returns "Yes" or a counter example for universal queries and "No" or a complete set of answers for existential queries. The last column depicts the number of found answers in the case of an existential query. For an universal query it shows either counter example (c.e.) or true.

$$Q_1 = \exists x(\text{politician}(x) \wedge \text{physicist}(x))$$

The query Q_1 is a conjunction of two sort atoms. This shows that reasoning about sorts and subsort relations is efficient. Note, that answering this query requires a reasoning about the sort theory of the ontology, in particular, the subsort relations.

$$Q_2 = \exists x, y, z(\text{hasSuccessor}(x, \text{GeorgeWBush}) \wedge \text{graduatedFrom}(x, z) \wedge \\ \text{graduatedFrom}(y, z) \wedge \text{isMarriedTo}(x, y))$$

The query Q_2 is a simple conjunction of predicates involving reasoning about the transitive predicate hasSuccessor. Figure 8.5 shows that the query answering procedures needs almost no time to answer this query.

$$Q_3 = \exists x, y(\text{bornIn}(\text{Angela\_Merkel}, y) \wedge \text{locatedIn}(x, y) \wedge \text{country}(y))$$

Query Q_3 involves reasoning about the transitive predicate locatedIn. The query answering procedure of SPASS–Y2 spends almost one minute for answering this query. This is because it needs to check for each country in the YAGO++ ontology if Hamburg, which is the birthplace of Angela Merkel, is located in this country. Each of these steps may require the execution of several chaining applications.

$$Q_4 = \exists x, y(\text{bornIn}(x, y) \wedge \forall z.\, \text{hasChild}(x, z) \rightarrow \text{bornIn}(z, y))$$

Although having many answers, the query Q_4, which contains a quantifier alternation, can be answered fast.

$$Q_5 = \exists x, y, v(\text{bornIn}(x, y) \wedge \text{physicist}(x) \ \wedge \forall z(\text{hasChild}(x, z) \rightarrow \text{bornIn}(z, y)))$$

This query is almost the same as the query Q_4, but further restricts the answers by the predicate physicist. This is the reason why fewer instances of the universal subquery have to be considered and, as a result, answering this query is faster than answering query Q_4.

$$Q_6 = \exists x, y, v(\text{bornIn}(x, y) \wedge \text{physicist}(x) \wedge \text{hasChild}(x, v) \wedge \\ \forall z(\text{hasChild}(x, z) \rightarrow \text{bornIn}(z, y)))$$

The query Q_6 is a restriction of query Q_5 by considering only those physicists who have at least one child. There are only two answers for query Q_6 in YAGO++ because the universal subquery of Q_4 and Q_5 is trivially fulfilled for all those physicists who do not have children.

$$Q_7 = \exists x(\text{bornIn}(x, y) \wedge \text{politician}(x) \wedge \text{locatedIn}(y, \text{Europe}) \wedge \text{physicist}(x))$$

Like query Q_3, the query Q_7 requires reasoning about the transitive predicate locatedIn. However, this query can be answered faster than query Q_3 because less reasoning about the transitive predicate is necessary. This is because the search space is further restricted by the predicate politician and the fact that the query answering procedure only needs to consider locations in Europe.

$$\begin{aligned} Q_8 = \exists x(&\text{bornIn}(x, \text{Hamburg}) \wedge \text{politician}(x) \wedge \text{physicist}(x) \wedge \\ &\text{hasSuccessor}(\text{Helmut\_Schmidt}, x)) \end{aligned}$$

Although hasSuccessor is a transitive predicate, answering this query is fast. After reasoning about the non-transitive predicates, the query answering procedure needs to perform just one reasoning task involving transitivity. This is because the YAGO++ ontology contains only one person who is born in Hamburg, is politician and physicist: Angela Merkel.

$$\begin{aligned} Q_9 = \forall x(&\text{politicianOf}(x, \text{Germany}) \rightarrow \exists y, z.\, \text{hasSuccessor}(y, x) \wedge \text{bornIn}(y, z) \wedge \\ &\text{locatedIn}(z, \text{Germany})) \end{aligned}$$

The query Q_9 is a quantifier alternation with an outermost universal quantifier. The query answering procedure of SPASS–Y2 finds a counterexample in almost no time.

$$Q_{10} = \exists x(\text{politician}(x) \wedge \text{bornInCountry}(x, \text{Germany}))$$

The predicate bornInCountry of query Q_{10} is a defined predicate which is transitive dependent (Figure 8.1). So, answering this query requires the query answering procedure to reason about the transitive predicate locatedIn.

$$Q_{11} = \forall x(\text{actress}(x) \rightarrow \text{performer}(x))$$

The query Q_{11} queries the minimal model of the YAGO++ ontology whether the property holds that every actress is also a performer. This holds for YAGO++, and SPASS-Y2 returns true in almost no time.

$$Q_{12} = \exists x(\text{GermanChancellor}(x) \wedge \forall y(\text{GermanChancellor}(y) \wedge \text{hasPredecessor}(x, y) \rightarrow \text{bornIn}(\text{AlbertEinstein}, \text{AlbertEinstein})))$$

The query Q_{12} uses the fact that the query language can express a restricted form of negation. It asks for the first German chancellor, i.e., the German chancellor who had no predecessor. The atom $\text{bornIn}(\text{AlbertEinstein}, \text{AlbertEinstein})$ does not hold in the minimal model of the YAGO++ ontology and, consequently, expresses $\bot$. As Figure 8.5 depicts, SPASS–Y2 also computes the answer for this query in almost no time.

query	time	#answers
Q_1	00.85	41
Q_2	00.00	1
Q_3	55.00	4
Q_4	03.15	35664
Q_5	00.56	779
Q_6	00.09	2
Q_7	06.71	3
Q_8	00.04	1
Q_9	00.13	c.e.
Q_{10}	23.94	73
Q_{11}	00.05	true
Q_{12}	00.01	1

Figure 8.5.: Query answering results: The table depicts the query answering times for each query. The time is given in the format seconds.milliseconds. For an existential query the table depicts the number of results and for an universal query either counterexample (c.e.) or true.

8.4. Summary

This chapter evaluates SPASS–Y2 that is based on SPASS and that implements the procedures presented in this thesis. SPASS–Y2 decides the consistency of YAGO++ within 16 minutes, the consistency of SUMO-Y2 in 53 minutes, and it finds inconsistencies of CYC-Y2 within one minute. The results depicted in Figure 8.3 show that SPASS-Y2 is the only system deciding the satisfiability of all three ontologies.

SPASS-Y2 is also the first superposition based query answering procedures that formally, completely, and efficiently answers, with respect to minimal model semantics, complex first-order queries, which contain arbitrary quantifier alternations, in a BSH-Y2 ontology that consists of several million clauses. The results depicted in Figure 8.5 show that this procedure answers queries with respect to minimal model semantics in the range of seconds. As a consequence, this is a practically useful query answering procedure.

SPASS–Y2 together with YAGO++, SUMO-Y2, CYC-Y2 and the queries are available from the SPASS homepage `http://www.spass-prover.org/` in section prototypes and experiments. There is also a prototype of a web frontend accessible from `http://spassyago.spass-prover.org/`.

9. Conclusion and Future Work

9.1. Robustness, Scalability and Usability

SPASS–Y2 implements the first superposition based reasoning procedures that efficiently decides the satisfiability of ontologies consisting of several million BSH-Y2 axioms. The set BSH-Y2 is a subset of the Bernays–Schönfinkel Horn fragment with equality. It is able to represent the YAGO ontology as well as large parts of the ontologies SUMO (SUMO-Y2) and CYC (CYC-Y2). In general, verifying the satisfiability in the Bernays–Schönfinkel Horn fragment is EXPTIME complete.

Additionally, SPASS–Y2 is the first efficient, sound, and complete reasoning procedure which decides the entailment of complex formulas in terms of minimal model semantics. This procedure performs an entailment check also for formulas containing arbitrary quantifier alternations in an ontology with several million formulas.

Therefore, the results of this work open several directions for future research which can basically be divided into three main directions of future investigation. The first direction is towards a parallelization of the methods presented in this thesis in order to lift them to Internet scale reasoning procedures distributed over several computers. The second direction is the development of a natural language style user interface which provides the query answering procedure to the non-expert Internet users. The last direction is towards more expressive languages providing a mechanism to express more information that can be stored in a knowledge base and used for answering more complex queries.

9.2. Parallelized Reasoning Procedures

In order to obtain reasoning procedures for the scale of the Internet, the reasoning has to be distributed over several computers. In order to gain also an additional constant speedup several algorithms such as the retrieval from the term index and the query answering procedure could possibly be parallelized.

In addition, also the saturation procedure can be parallelized by a stratification [SPT11] of the input clause set. Because of the fact that there are no cycles in the BSH-Y2 there is always a stratification. A stratification gives a layering of the original input showing which clauses depend from other clauses. This identifies independent parts of a clause set, and the layering defines a saturation ordering, i.e., which parts of a clause set have to be saturated before other parts.

The previous version of SPASS–Y2 that I have presented in [SWW10] is called SPASS-YAGO. An execution of several parallel instances of SPASS-YAGO, which exchange information via a network protocol, is examined in [Sch12]. The clause set representing the logical representation is split among the SPASS-YAGO instances. The experiments show that this also splits the saturation work among the instances. Similar ideas could be possibly used to also split query answering among several instances of the query answering procedure.

9.3. Natural language interface

In its current status, SPASS–Y2 is only suitable for usage by experts because the queries have to be formulated in first-order logic. A natural language interface, which is can generate the first-order queries from natural language queries, is the missing link. A natural language interface would make the query answering procedure of SPASS–Y2 available to a broader number of users.

9.4. Going beyond BSH-Y2

The current version of YAGO which is YAGO 2 [HSB+11] contains constructs that I have not explicitly considered in this work. These are confidence values, time and location information. Further investigations could be done by examining how the procedure presented in this thesis could be appropriately adapted to support also to perform reasoning about this information.

Many special purpose ontologies are represented in a description logic. Because of the fact that description logics are decidable fragments of first-order logic, it would be interesting to further investigate the question whether the reasoning procedures of SPASS–Y2 can be adapted such that they also support description logic languages.

9.4.1. Reasoning with Confidences

Attaching each fact with a confidence value allows a reasoning procedure to relativize the facts among each other. These values also help to resolve a conflict occurring in a clause set representing a particular knowledge base. If information is automatically extracted from different sources then these sources might have contrary information. A confidence value attached with the extracted facts can possibly resolve the conflict.

Consider the following clause set depicting this problem. A clause with a confidence attached is a tuple (C, α) where C is a clause and α is a confidence value with $0 \leq \alpha \leq 1$.

$$(\text{bornIn}(\text{AlbertEinstein}, \text{Ulm}), 0.9) \tag{9.1}$$
$$(\text{bornIn}(\text{AlbertEinstein}, \text{Munich}), 0.1) \tag{9.2}$$
$$(\neg\, \text{bornIn}(x, y) \vee \neg\, \text{bornIn}(x, z) \vee y \approx z, 1.0) \tag{9.3}$$
$$(\text{Ulm} \not\approx \text{Munich}, 1.0) \tag{9.4}$$

Ignoring the confidence values, we see that this clause set is unsatisfiable. But the clause 9.2 has a very low confidence measure and this gives evidence that clause 9.1 is true and clause 9.2 false. The following resolution rule gives an example of an inference respecting the confidence values.

The value α is called a confidence measure if $0 \leq \alpha \leq 1$. Let L_i and L_j be literals for $i \in \{1, \ldots n\}$, $j \in \{1, \ldots, m\}$, A be an atom, and α, β be confidence measures then the following inference is a resolution with confidences [DLP94].

$$\frac{(L_1 \vee \cdots \vee L_n \vee A, \alpha) \quad (L'_1 \vee \cdots \vee L'_m \vee \neg B, \beta)}{((L_1, \vee \cdots \vee L_n \vee L'_1, \vee \cdots \vee L'_m)\sigma, \min(\alpha, \beta))}$$

if σ is the most general unifier of A and B.

Resolution between Clause 9.1 and Clause 9.2 infers the following clause:

$$(\text{bornIn}(\text{AlbertEinstein}, z) \vee \text{Ulm} \approx z, 0.9). \tag{9.5}$$

Performing a resolution between Clause 9.2 and Clause 9.5 infers

$$(\text{Ulm} \approx \text{Munich}, 0.1), \tag{9.6}$$

and resolution between Clause 9.6 and Clause 9.4 derives

$$(\text{false}, 0.1). \tag{9.7}$$

This example derives false from the Clauses 9.1–9.4 with confidence 0.1. Without considering the confidences the resolution calculus would derive false. As a consequence, this clause set is inconsistent and collapses. However, considering the attached confidence values, one could conclude that bornIn(AlbertEinstein, Ulm) holds because this fact has a higher confidence than the conflict false, clause 9.7.

9.4.2. Reasoning with Arithmetic

The ontology YAGO 2 [HSB+11], which is the successor of the YAGO ontology, has additional information about time and location attached with facts. With this additional information queries can be answered involving time and location aspects. Reasoning about time and locations, requires particular calculi that are able to reason about arithmetic constraints.

The query "Which physicist survived all his children" can be formulated in first-order logic with arithmetic constraints as follows

$$\begin{aligned}\exists x, d_1(\text{physicist}(x) \wedge \text{diedIn}(x, d_1) \wedge \\ \forall z, d_2(\text{hasChild}(x, z) \wedge \text{diedIn}(\text{z}, \text{d}_2) \rightarrow d_1 > d_2))\end{aligned} \tag{9.8}$$

Assume a saturated clause set N of a translation of YAGO 2 with the same properties as the saturation of a clause set from the BSH-Y2. Assume further, the clause set contains the following clauses

$$\rightarrow \text{physicist}(\text{MaxPlanck}) \tag{9.9}$$
$$\rightarrow \text{diedInYear}(\text{MaxPlanck}, 1947) \tag{9.10}$$

$$\rightarrow \text{hasChild}(\text{MaxPlanck}, \text{KarlPlanck}) \tag{9.11}$$
$$\rightarrow \text{hasChild}(\text{MaxPlanck}, \text{EmmaPlanck}) \tag{9.12}$$
$$\rightarrow \text{hasChild}(\text{MaxPlanck}, \text{GretePlanck}) \tag{9.13}$$
$$\rightarrow \text{hasChild}(\text{MaxPlanck}, \text{ErwinPlanck}) \tag{9.14}$$

$$\rightarrow \text{diedIn}(\text{KarlPlanck}, 1916) \tag{9.15}$$
$$\rightarrow \text{diedIn}(\text{EmmaPlanck}, 1919) \tag{9.16}$$
$$\rightarrow \text{diedIn}(\text{GretePlanck}, 1917) \tag{9.17}$$
$$\rightarrow \text{diedIn}(\text{ErwinPlanck}, 1945) \tag{9.18}$$

Similar to the query answering procedure of Chapter 7, instantiating the query 9.8 with these clauses would result in the following four constraints:

$$1947 > 1916 \tag{9.19}$$
$$1947 > 1919 \tag{9.20}$$
$$1947 > 1917 \tag{9.21}$$
$$1947 > 1945 \tag{9.22}$$

These constraints are ground and can be decided [KW11]. Since, all of these constraints are fulfilled, MaxPlanck is an answer for query 9.8. It remains for future investigation to verify if the approach, presented in this thesis, can indeed be extended accordingly while remaining sound and complete.

The YAGO 2 ontology contains also locational information in the form of geographical coordinates. Reasoning about this information requires a calculus also working on arithmetic constraints. Then also queries involving locational specifications, for example, north of or within 10 kilometers, would be possible; for example "What is the southernmost city of Europe?"

$$\begin{aligned}\exists x, l_1(\text{city}(x) \wedge \text{locatedIn}(x, \text{Europe}) \wedge \text{latitudeOf}(x, l_1) \wedge \\ \forall y, l_2(\text{city}(y) \wedge \text{locatedIn}(y, \text{Europe}) \wedge \text{latitudeOf}(y, l_2) \rightarrow l_2 > l_1))\end{aligned}$$

9.4.3. Reasoning in Description Logics

Description logics like the standard description logic $\mathcal{ALC}$ and DL–Lite are decidable [BCM+03]. These languages allow an existential quantified variable occurring in a clause. For example, this allows the formulation of constraints like "Every country has a capital city" which is in first-order notations

$$\forall x(\text{country}(x) \rightarrow \exists y(\text{hasCapital}(x, y))) \tag{9.23}$$

The paper [CGL09] shows that Datalog [CGT89] can be extended with this construct. This extension is decidable if the respective clauses contain a guard. A guard is an atom that contains all non-existential quantified variables.

The work, presented in [GdN99], shows that superposition is a decision procedure for the guarded fragment of first-order logic. It remains to verify if the procedures of this thesis can be appropriately adapted using these results in order to also decide clause sets from BSH-Y2 plus clauses like clause 9.23.

9.4.4. Higher-Order Queries

Asking the YAGO ontology for the relation that two entities have in common, seems to be a higher-order query. If we consider finite domain reasoning this higher-order construct can be replaced by a finite disjunction over first-order clauses.

For example consider a clause set from the BSH-Y2 and the following query:

$$\exists x, F(F(\text{AlbertEinstein}, x) \ \wedge \ F(\text{MaxPlanck}, x)), \tag{9.24}$$

where F is a variable for a predicate symbol. Let $\text{preds}(N)$ be the set of all predicate symbols occurring in N. In the case of finite domain reasoning, the query 9.24 can be rewritten as a finite disjunction as follows

$$\exists x \left(\bigvee_{P \in \text{preds}(N)} P(\text{AlbertEinstein}, x) \ \wedge \ P(\text{MaxPlanck}, x) \right) \tag{9.25}$$

The filtered context tree term index, presented in Chapter 5, is able to efficiently retrieve the respective instances from the index as mentioned in Section 5.4. Consequently, the query answering procedure could be extended respectively. Additionally, this also requires that the filtered context tree index is implemented in SPASS–Y2. The current implementation of SPASS–Y2 contains only an implementation of the filtered substitution tree index that is a special case of the filtered context tree index. The substitution tree index does not support variables representing predicate symbols.

9.4.5. Extended Query Language

The query language could be extended in such a way that more than one subquery is allowed. The new syntax looks as follows:

$$\Phi \quad := \quad \Gamma \quad | \quad \forall x(\Gamma \wedge \bigwedge \Phi \rightarrow \Phi) \quad | \quad \exists x(\Gamma \wedge \bigwedge \Phi \wedge \Phi) \quad | \quad \top$$

where $\bigwedge \Phi$ could possibly be an empty disjunction. If the shielding property (Definition 80) of queries is appropriately adapted the query answering procedure of Chapter 7 could possibly be adapted to also answer this type of queries.

Part II.

Reductions for Automated Theorem Proving

10. Introduction

In the superposition context, first-order theorem proving with equality deals with the problem of showing unsatisfiability of a finite set N of clauses. This problem is well-known to be undecidable, in general. It is semi-decidable in the sense that superposition is refutationally complete. As shown in Chapter 2, the superposition reasoning framework is composed of inference and reduction rules. Inference rules generate new clauses from N whereas reduction rules delete clauses from N or transform them into simpler ones while deleting the ancestors. If, in particular, powerful reduction rules are available, decidability of certain subclasses of first-order logic can be shown and explored in practice [BGW93, HSG04, JMW98, GdN99, FLHT01]. Hence, sophisticated reductions are an important means for progress in automated theorem proving. In this work I have developed an instance of the reduction rule *contextual rewriting* called *subterm contextual rewriting* which is considered in combination with the superposition calculus [BG94]. Contextual rewriting extends rewriting with unit equations to rewriting with full clauses containing a positive orientable equation. In order to apply such a clause C for rewriting a clause D, all other literals of C have to be entailed by the literals of D in N. The literals of D are called the context. The name contextual rewriting comes from the inclusion of this context. The instance subterm contextual rewriting introduced in this work, restricts the instantiation of the context to subterms of the involved potentially rewritten clause.

For a first, simple example consider the two clauses

$$P(x) \to f(x) \approx x \tag{10.1}$$

$$S(g(a)), a \approx b, P(b) \to R(f(a)) \tag{10.2}$$

Clauses are written in implication form [Wei01]. A brief introduction of this is also provided in Chapter 2. Now in order to rewrite $R(f(a))$ in clause 10.2 to $R(a)$ using the equation $f(x) \approx x$ of clause 10.1 with matcher $\sigma = \{x \mapsto a\}$, it needs to be shown that $P(x)\sigma$ holds in the context of clause 10.2 $S(g(a)), a \approx b, P(b)$, i.e., $\models S(g(a)), a \approx b, P(b) \to P(x)\sigma$. This obviously holds, so a contextual rewriting application of clause 10.1 can replace clause 10.2 by $S(g(a)), a \approx b, P(b) \to R(a)$.

More generally, contextual rewriting is the following reduction rule:

$$\mathcal{R}\frac{D = \Gamma_1 \to \Delta_1, s \approx t \quad C = (\Gamma_2 \to \Delta_2)[u[s\sigma] \approx v]}{\begin{array}{c}\Gamma_1 \to \Delta_1, s \approx t \\ (\Gamma_2 \to \Delta_2)[u[t\sigma] \approx v]\end{array}}$$

where $(\Gamma_2 \to \Delta_2)[u[s\sigma] \approx v]$ expresses that $u[s\sigma] \approx v$ is an atom occurring in Γ_2 or Δ_2 and u contains the subterm $s\sigma$. Contextual rewriting reduces the subterm $s\sigma$ of u to $t\sigma$ if, apart from additional ordering restrictions, the following conditions are satisfied

$$N_C \models \Gamma_2 \to A \text{ for all } A \text{ in } \Gamma_1\sigma \tag{10.3}$$
$$N_C \models A \to \Delta_2 \text{ for all } A \text{ in } \Delta_1\sigma \tag{10.4}$$

where N is the current clause set, $C, D \in N$, and N_C denotes the set of clauses from N smaller than C with respect to an ordering $\prec$, total on ground terms (Section 2.2.2). All calculus rules considered in this thesis are actually reduction rules (Section 2.2.4). Both side conditions 10.3 and 10.4 are undecidable, in general. Therefore, in order to make the contextual rewriting rule applicable in practice, it must be instantiated such that eventually these two conditions become effectively decidable.

10.1. Example

For a more sophisticated, further motivating example, consider the following clause set. It can be finitely saturated using contextual rewriting but not solely with less sophisticated reduction mechanisms such as unit rewriting or subsumption. Consider the following inference superposition right [BG90, Wei01].

Superposition right

$$\frac{\Gamma_1 \to \Delta_1, l \approx r \quad \Gamma_2 \to \Delta_2, s[l']_p \approx t}{(\Gamma_1, \Gamma_2 \to \Delta_1, \Delta_2, s[r]_p \approx t)\sigma},$$

where (i) σ is the most general unifier of l' and l, (ii) l' is not a variable, (iii) no literal in Γ_1, Γ_2 is selected, (iv) $r\sigma \not\succ l\sigma$, (v) $l\sigma \approx r\sigma$ is reductive for $(\Gamma_1 \to \Delta_1, l \approx r)\sigma$, and (vi) $s\sigma \approx t\sigma$ is reductive for $(\Gamma_2 \to \Delta_2, s[l']_p \approx t)$

Let i, q, r, f be functions, a, b, n be constants and x_1, x_2, x_3, x_4, y_1 be variables and the precedence set as follows $r > f > q > i > b > a > n$ using the KBO (Section 2.2.2) with weight 1 for all function symbols and variables.

$$\rightarrow q(n) \approx b \tag{10.5}$$

$$i(x_1) \approx b, q(y_1) \approx b \rightarrow q(r(x_1, y_1)) \approx b \tag{10.6}$$

$$i(x_1) \approx b, q(y_1) \approx b \rightarrow q(f(x_1, y_1)) \approx a \tag{10.7}$$

$$\begin{aligned} i(x_1) \approx b, q(y_1) \approx b, & i(x_3) \approx b \rightarrow \\ & r(x_3, f(x_1, y_1)) \approx f(x_1, r(x_3, y_1)) \end{aligned} \tag{10.8}$$

$$\begin{aligned} i(x_1) \approx b, i(x_3) \approx b, & i(x_2) \approx b, \\ & q(y_1) \approx b, b \approx a \rightarrow \\ y_1 \approx n, & q(f(x_1, f(x_2, r(x_3, y_1)))) \approx b \end{aligned} \tag{10.9}$$

If we apply superposition right between clause 10.8 and clause 10.9 on the term $q(f(x_1, f(x_2, r(x_3, y_1))))$ we obtain the clause

$$\begin{aligned} i(x_1) \approx b, i(x_3) \approx b, & i(x_2) \approx b, i(x_4) \approx b, \\ & q(y_1) \approx b, q(f(x_4, y_1)) \approx b, b \approx a \\ & \rightarrow \\ f(x_4, y_1) \approx n, & \\ & q(f(x_1, f(x_2, f(x_4, r(x_3, y_1))))) \approx b \end{aligned} \tag{10.10}$$

which is larger (both in the ordering and the number of symbols) than clause 10.9. Applying superposition between clause 10.8 and clause 10.10 yields an even larger clause. Repeating the superposition inference between clause 10.8 and these clauses creates larger and larger clauses. All those clauses cannot be simplified by unit rewriting, non-unit rewriting and are not redundant with respect to subsumption deletion [Wei01] (Chapter 2.2.4). Hence, the exhaustive application of the superposition calculus does not terminate on this clause set. However, contextual rewriting can reduce clause 10.9 using clause 10.7 to

$$\begin{aligned} i(x_1) \approx b, i(x_3) \approx b, & i(x_2) \approx b, \\ & q(y_1) \approx b, b \approx a \rightarrow y_1 \approx n, a \approx b. \end{aligned} \tag{10.11}$$

Clause 10.11 is a tautology and can be reduced to true. Then the set is saturated since no further superposition inference is possible. In order to apply contextual rewriting to clause 10.9 using clause 10.7 the following two side conditions have to be verified

$$N_C \models i(x_1) \approx b, i(x_3) \approx b, i(x_2) \approx b, \\ q(y_1) \approx b, b \approx a \to i(x_1) \approx b$$

and

$$N_C \models i(x_1) \approx b, i(x_3) \approx b, i(x_2) \approx b, q(y_1) \approx b, \\ b \approx a \to q(f(x_2, r(x_3, y_1))) \approx b.$$

The first condition holds trivially and the latter follows from clause 10.7 and clause 10.6 which are both smaller than clause 10.9. The example will be explained in full detail in Chapter 13. It already shows that the class of clause sets that can be finitely saturated with contextual rewriting is strictly larger than the class of clause sets that can be finitely saturated by unit rewriting, non-unit rewriting or local contextual rewriting [Wei01, WBH+02].

For the superposition calculus, contextual rewriting was first implemented in the SATURATE system [NN93, GN94] but never matured. On one hand it turned out to be indispensable for proving a number of examples, on the other hand the implementation was not able to decide even a single rule application in reasonable time for a bunch of other examples. This was partly due to a straightforward naive implementation, compared to the techniques developed in this work, and a more general setting where the ordering constraints of the rule were not a priori calculated, but inherited through ordering constraints.

10.2. Related work

Apart from the superposition calculus, contextual rewriting was already intensively studied for purely equational (Horn) rewrite systems and in the context of inductive theorem proving [Gan87, Zha93, BR95]. Comparing the variants of contextual rewriting defined there, they are less general with respect to non-Horn clauses but also less restrictive with respect to ordering restrictions. However, the ordering restrictions imposed on the superposition variant of the rule, Definition 85, are needed to preserve completeness of the superposition calculus. In an inductive theorem proving setting, completeness is typically not an issue as it cannot be obtained anyway.

10.3. Contribution

In this part of the thesis, I present a decidable instance of contextual rewriting, called subterm contextual rewriting, that enables the superposition reasoning procedure to

find proofs in hard theories. Furthermore, it increases the number of first-order problems for which superposition is a decision procedure. Subterm contextual rewriting is presented in Chapter 11. I have implemented subterm contextual rewriting in SPASS 3.1. [WBH+02] (Chapter 12) and tested this implementation on all problems of the *TPTP* library version 3.2.0 [SS98] (Chapter 13). Compared to my first implementation of the rule [WW08] the results of this work lead to significant increase in performance. The extended and refined implementation wins significantly more problems on the overall TPTP than it loses while keeping the positive results on hard problems. In particular, it solves 6 problems from the TPTP that no other reported system could solve before. The gained performance is due to a tight incorporation of contextual rewriting with unit and non-unit rewriting and a new caching technique, see Chapter 12. Finally, in Chapter 14, I prove that subterm contextual rewriting is a decision procedure for ground equations in the minimal model of a universally reductive equational theory. Subterm contextual rewriting is the first effective decision procedure for this problem. I have the presented work already published in [WW10].

11. Subterm Contextual Rewriting

This section presents the subterm contextual rewriting rule, Definition 88, which is a decidable instance of the general superposition variant of contextual rewriting, Definition 85. To this end I have developed side conditions of contextual rewriting which are decidable and feasible for problems occurring in practice. This is done by restricting the considered instances for the context to ground subterms of the rewritten clause, Section 11.2. In addition, the testing of these instances is restricted by a particular subterm contextual rewriting rule for ground clauses, Definition 87.

In addition to the standard first-order reasoning framework presented in Chapter 2 this part requires two further notions. A clause C is called *reductive* for a positive equation $s \approx t$ if $s \approx t$ is strictly maximal in C and $s \succ t$. Furthermore, C is called *universally reductive*[GS93], if in addition $\mathrm{vars}(s) \supseteq \mathrm{vars}(C)$.

11.1. Contextual Rewriting

Contextual rewriting is a sophisticated reduction rule originally introduced in [BG94] that generalizes unit rewriting and non-unit rewriting [Wei01]. It is an instance of the standard redundancy notion of superposition as shown in Section 2.2.4.

Definition 85 (Contextual Rewriting [BG94])**.** *Let N be a clause set, $C, D \in N$, σ be a substitution then the reductions*

$$\mathcal{R}\frac{D = \Gamma_1 \to \Delta_1, s \approx t \quad C = \Gamma_2, u[s\sigma] \approx v \to \Delta_2}{\begin{array}{c}\Gamma_1 \to \Delta_1, s \approx t \\ C' = \Gamma_2, u[t\sigma] \approx v \to \Delta_2\end{array}}$$

$$\mathcal{R}\frac{D = \Gamma_1 \to \Delta_1, s \approx t \quad C = \Gamma_2 \to \Delta_2, u[s\sigma] \approx v}{\begin{array}{c}\Gamma_1 \to \Delta_1, s \approx t \\ C' = \Gamma_2 \to \Delta_2, u[t\sigma] \approx v\end{array}}$$

where the following conditions are satisfied

1. $s\sigma \succ t\sigma$
2. $C \succ D\sigma$
3. $N_C \models \Gamma_2 \rightarrow A$ *for all* A *in* $\Gamma_1\sigma$
4. $N_C \models A \rightarrow \Delta_2$ *for all* A *in* $\Delta_1\sigma$

are called contextual rewriting.

Due to condition 85-1 and condition 85-2, $C' \prec C$ and $D\sigma \prec C$. Then from condition 85-3 and condition 85-4, it follows that there exist clauses $C_1, \ldots, C_n \in N_C$ and $C_1, \ldots, C_n, C', D\sigma \models C$. Therefore, the clause C is redundant in $N \cup \{C'\}$ and can be replaced by C'. The rule is an instance of the abstract superposition redundancy notion (Section 2.2.4).

The side conditions 85-3 and 85-4 having both the form $N_C \models \Gamma \rightarrow \Delta$ are undecidable, in general. There are two sources for undecidability of conditions having the form $N_C \models \Gamma \rightarrow \Delta$. First, there are infinitely many possible grounding substitutions for the clauses $\Gamma \rightarrow \Delta$ and C. Second, even for a given grounding substitution σ' there may be infinitely many ground substitutions δ with $C_i\delta \prec C\sigma'$, $C_i \in N$, e.g., if $\prec$ is the lexicographic path ordering (LPO, Section 2.2.2). Therefore, in order to effectively decide the side conditions, the following approach fixes one σ' and restricts δ such that $\mathrm{cod}(\delta)$ is a subset of the subterms from C. This yields subterm contextual rewriting.

11.2. Developing Feasible Side Conditions

First, $N_C \models \Gamma \rightarrow \Delta$ is equivalent to $N_C \cup \{\exists x_1, \ldots, x_n.\neg(\Gamma \rightarrow \Delta)\} \models \bot$ where the x_i are the variables of $\Gamma \rightarrow \Delta$. The existential quantifier can be eliminated by Skolemization yielding a Skolem substitution τ that maps any x_i to a new Skolem constant. Consequently, setting σ' to τ yields the instance $N_C \models (\Gamma \rightarrow \Delta)\tau$, where $(\Gamma \rightarrow \Delta)\tau$ is ground. Still there may exist infinitely many δ with $C_i\delta \prec C\tau$, $C_i \in N$. Furthermore, $C\tau$ may still contain variables as the literal $u[t\sigma] \approx v$ of C may contain variables that do not occur in Γ_2, Δ_2.

Therefore, I restrict δ to those grounding substitutions that map variables to terms only occurring in $C\tau$ or $D\sigma\tau$ where I additionally assume that τ is also grounding for C and $D\sigma$, i.e., it maps any variable occurring in C or $D\sigma$ to an arbitrary fresh Skolem constant. Let $N_{C\tau}^{D\sigma\tau}$ be the set of all ground instances of clauses from N smaller than $C\tau$ obtained by instantiation with ground terms from $D\sigma\tau, C\tau$. Then $N_{C\tau}^{D\sigma\tau}$ is finite and $N_{C\tau}^{D\sigma\tau} \subseteq N_{C\tau}$. Consequently, $N_{C\tau}^{D\sigma\tau} \models (\Gamma \rightarrow \Delta)\tau$ is a sufficient ground approximation of $N_C \models \Gamma \rightarrow \Delta$.

Even though this is a decidable approximation of the original problem the set $N_{C\tau}^{D\sigma\tau}$ is exponentially larger than N, in general. In particular, the set typically already gets so large that an instantiation based theorem proving approach does not always work out

deciding $N_{C\tau}^{D\sigma\tau} \models (\Gamma \to \Delta)\tau$. For example, the rewriting step from the example in the introduction contains already more than 20 different ground terms out of

$$\begin{gathered} i(c_1) \approx b, i(c_3) \approx b, i(c_2) \approx b, \\ i(c_4) \approx b, q(c_5) \approx b, q(f(c_4, c_5)) \approx b, b \approx a \\ \to \\ f(c_4, c_5) \approx n, q(f(c_1, f(c_2, f(c_4, r(c_3, c_5))))) \approx b \end{gathered}$$

where the c_i are the freshly introduced Skolem constants. Recall that N is not the input clause set but the set of all clauses generated in the course of a saturation and can thus consists of several (hundred) thousand clauses. The side condition $N_{C\tau}^{D\sigma\tau} \models (\Gamma \to \Delta)\tau$ is typically tested several 10 thousand times for a problem with potential contextual rewriting applications, even with respect to the refinements that are introduced in the sequel. Therefore, the following definition gives a redundancy notion that implicitly represents $N_{C\tau}^{D\sigma\tau}$ and approximates $N_{C\tau}^{D\sigma\tau} \models (\Gamma \to \Delta)\tau$.

Definition 86 (Ground subterm redundancy)**.** *A clause is ground subterm redundant, if it can be reduced to* $\top$ *by the reduction rules* tautology reduction*,* forward subsumption*,* obvious reduction *and a particular instance of contextual rewriting called* subterm contextual ground rewriting *defined below.*

Tautology reduction reduces syntactic and semantic tautologies to true whereas forward subsumption reduces subsumed clauses to true. Obvious reduction eliminates trivial literals [Wei01]. A procedure deciding ground subterm redundancy is shown in Algorithm 16 and explained in detail in Chapter 12.

Ground subterm redundancy only applies to ground clauses. Therefore, the following definition introduces an instance of contextual rewriting only working on ground clauses. Further, it refines contextual rewriting such that it implicitly only considers clauses from $N_{C\tau}^{D\sigma\tau}$. This is in particular guaranteed by condition 87-3 below that limits the clauses used for reductions to universally reductive clauses.

Definition 87 (Subterm Contextual Ground Rewriting)**.** *If* N *is a clause set,* $D \in N$*,* C' *ground,* σ *a substitution then the reductions*

$$\mathcal{R}\frac{D = \Gamma_1 \to \Delta_1, s \approx t \quad C' = \Gamma_2, u[s\sigma] \approx v \to \Delta_2}{\begin{gathered}\Gamma_1 \to \Delta_1, s \approx t \\ \Gamma_2, u[t\sigma] \approx v \to \Delta_2\end{gathered}}$$

$$\mathcal{R}\frac{D = \Gamma_1 \to \Delta_1, s \approx t \quad C' = \Gamma_2 \to \Delta_2, u[s\sigma] \approx v}{\begin{gathered}\Gamma_1 \to \Delta_1, s \approx t \\ \Gamma_2 \to \Delta_2, u[t\sigma] \approx v\end{gathered}}$$

where the following conditions are satisfied

1. $s\sigma$ *is a strictly maximal term in* $D\sigma$
2. $u[s\sigma] \approx v \succ s\sigma \approx t\sigma$
3. $\mathrm{vars}(s) \supseteq \mathrm{vars}(D)$
4. $(\Gamma_2 \to A)$ *is ground subterm redundant for all* A *in* $\Gamma_1\sigma$
5. $(A \to \Delta_2)$ *is ground subterm redundant for all* A *in* $\Delta_1\sigma$

are called subterm contextual ground rewriting.

Condition 87-1 and condition 87-2 ensure the ordering restrictions required by contextual rewriting. Condition 87-3 implies that $D\sigma$ is ground. Recall that any clause meeting condition 87-1 and condition 87-3 is universally reductive. Condition 87-4 and condition 87-5 recursively apply the ground subterm redundancy criterion.

The ground subterm redundancy criterion is terminating since C' is reduced to a smaller ground clause. As a consequence, also the ground subterm redundancy procedure (Algorithm 16) is terminating.

11.3. Subterm Contextual Rewriting

The instance of contextual rewriting, namely the subterm contextual rewriting rule, becomes the below reduction rule.

Definition 88 (Subterm Contextual Rewriting)**.** *Let* N *be a clause set,* $C, D \in N$*,* σ *be a substitution then the reductions*

$$\mathcal{R}\frac{D = \Gamma_1 \to \Delta_1, s \approx t \quad C = \Gamma_2, u[s\sigma] \approx v \to \Delta_2}{\begin{array}{c}\Gamma_1 \to \Delta_1, s \approx t \\ \Gamma_2, u[t\sigma] \approx v \to \Delta_2\end{array}}$$

$$\mathcal{R}\frac{D = \Gamma_1 \to \Delta_1, s \approx t \quad C = \Gamma_2 \to \Delta_2, u[s\sigma] \approx v}{\begin{array}{c}\Gamma_1 \to \Delta_1, s \approx t \\ \Gamma_2 \to \Delta_2, u[t\sigma] \approx v\end{array}}$$

where the following conditions are satisfied

1. $s\sigma \succ t\sigma$

2. $C \succ D\sigma$
3. τ *maps all variables from* $C, D\sigma$ *to fresh Skolem constants*
4. $(\Gamma_2 \to A)\tau$ *is ground subterm redundant for all* A *in* $\Gamma_1\sigma$
5. $(A \to \Delta_2)\tau$ *is ground subterm redundant for all* A *in* $\Delta_1\sigma$

are called subterm contextual rewriting.

Note that unit rewriting and non-unit rewriting [Wei01] are also instances of the subterm contextual rewriting rule. Note further that the conditions for the subterm contextual rewriting rule are weaker compared to the subterm contextual ground rewriting rule: the right premise does not need to be ground and the equation $s \approx t$ needs not to be maximal in the first premise. Subterm contextual rewriting uses subterm contextual ground rewriting to effectively decide the side conditions.

In addition to the rewriting style, where subterms are replaced by simpler ones, the general idea of contextual rewriting can also be used to actually eliminate literals, resulting in a generalization of matching replacement resolution [Wei01]. This variant then also considers negative literals for reductions and was used for the experiments on the TPTP, Chapter 13.

Definition 89 (Subterm Contextual Literal Elimination)**.** *Let* N *be a clause set,* $C, D \in N$*,* σ *be a substitution then the reductions*

$$\mathcal{R}\frac{D = \Gamma_1 \to \Delta_1, s \approx t \quad C = \Gamma_2, u \approx v \to \Delta_2}{\begin{array}{c}\Gamma_1 \to \Delta_1, s \approx t \\ \Gamma_2 \to \Delta_2\end{array}}$$

$$\mathcal{R}\frac{D = \Gamma_1, s \approx t \to \Delta_1 \quad C = \Gamma_2 \to \Delta_2, u \approx v}{\begin{array}{c}\Gamma_1, s \approx t \to \Delta_1 \\ \Gamma_2 \to \Delta_2\end{array}}$$

where the following conditions are satisfied

1. $s\sigma = u$ *and* $t\sigma = v$
2. $C \succ D\sigma$
3. τ *maps all variables from* $C, D\sigma$ *to fresh Skolem constants*
4. $(\Gamma_2 \to A)\tau$ *is ground subterm redundant for all* A *in* $\Gamma_1\sigma$
5. $(A \to \Delta_2)\tau$ *is ground subterm redundant for all* A *in* $\Delta_1\sigma$

are called subterm contextual literal elimination.

12. Implementation

The implementation of SPASS [Wei01] focuses on a sophisticated reduction machinery. As shown in Section 2.2 the SPASS main loop performs an exhaustive application of its inference and reduction rules. The integration of contextual rewriting into the reduction procedure of SPASS consists of two steps: finding rewrite candidates and verifying the side conditions of subterm contextual rewriting. These two steps are presented in this chapter. In addition, further techniques are presented that increase the performance of the subterm contextual rewriting procedure on practical reasoning problems.

12.1. Finding Rewrite Candidates

First, I consider the search for appropriate contextual rewrite application candidates. This is analogous to the case of unit rewriting and non-unit rewriting. Finding appropriate rewrite candidates is realized in SPASS via substitution tree indexing [Gra96, NHRV01] which is an instance of context tree indexing [GNN01] shown in detail in Section 2.2.5. An index has the functionality of a database for automated theorem proving. It does not store relations, but terms, i.e. trees, and instead of the typical database operations it provides queries delivering terms with respect to the instance, generalization, and unifiable relation. The unifiable relation is used to find partners for inferences, the other two in order to find partners for the backward and forward application of reduction rules, respectively.

The following shows the non-unit rewriting rule as it is considered by several superposition based provers.

Definition 90 (Non-Unit Rewriting)**.**

$$\mathcal{R}\frac{D = \Gamma_1 \to \Delta_1, s \approx t \quad C = \Gamma_2, u[s\sigma] \approx v \to \Delta_2}{\begin{array}{c}\Gamma_1 \to \Delta_1, s \approx t \\ C'' = \Gamma_2, u[t\sigma] \approx v \to \Delta_2\end{array}}$$

$$\mathcal{R}\frac{D = \Gamma_1 \to \Delta_1, s \approx t \quad C = \Gamma_2 \to \Delta_2, u[s\sigma] \approx v}{\begin{array}{c}\Gamma_1 \to \Delta_1, s \approx t \\ C'' = \Gamma_2 \to \Delta_2, u[t\sigma] \approx v\end{array}}$$

where (i) $s \succ t$ *and (ii)* $\Gamma_1\sigma \subset \Gamma_2, \Delta_1\sigma \subset \Delta_2$.

In order to forward rewrite a clause C the implementation of unit and non-unit rewriting tries to reduce each subterm $s\sigma$ of C. Therefore, for each subterm $s\sigma$ the procedure queries the substitution tree index for generalization, i.e. whether there exist a candidate term l with $l\delta = s\sigma$. With respect to the above rule context, $l = s$ and $\delta = \sigma$ would be one result of the query. If there exists such a term then the substitution tree index returns l together with the matcher δ. For all clauses D containing a generalization l, the requirements (i) and (ii) are verified. If they are fulfilled for a specific D, then C is rewritten. Otherwise, the implementation queries the substitution tree index for the next candidate term l. The retrieval is realized for forward reduction in an iterative way because the first hit is already used for performing the reduction and the remaining hits can be ignored.

12.2. Ground Subterm Redundancy Check

The second step for integrating contextual rewriting into SPASS is to check the side conditions that require an effective implementation of the ground subterm redundancy check.

First of all, it is too costly to explicitly compute the Skolem substitution τ for each clause $(\Gamma \rightarrow \Delta)\tau$ subject to the ground subterm redundancy criterion. Applying τ explicitly requires to allocate memory for the new constants, the resulting terms and the new clause and it requires additional computations to build the clause. Because of the recursive structure of the redundancy criterion this is not feasible. Therefore, my solution is to simply treat variables as constants in the implementation of the redundancy criterion.

In SPASS constants and variables in terms are represented by particular nodes. If the implementation of subterm contextual rewriting replaced the variables of the clause $\Gamma \rightarrow \Delta$ explicitly by fresh constants, then it would create for each variable a new node carrying a constant and inserting the new constant into the existing signature precedence with lowest precedence. Internally in SPASS, variables are represented by positive integers. So variables are implicitly ordered. This allows SPASS to consider variables to be constants with a lower precedence than any other non-variable symbol of the signature and take the integer ordering among the variables. Using this trick and adapting the ordering modules (KBO, RPOS) of SPASS such that they treat variables exactly as constants in the context of subterm contextual rewriting, this approach has the same properties with respect to ordering computation as creating constants explicitly but saves the additional effort and memory consumption.

If variables are interpreted as constants the standard procedure of SPASS for finding appropriate rewrite candidates can remain unchanged, because a fresh constant as well as a variable have only a variable as its generalization.

The following algorithm presents the implementation of the ground subterm redundancy check. It works exactly as an implementation that explicitly creates fresh Skolem constants.

Algorithm 16: GroundSubtermRedundant

```
   Input: clause set N, clause C
 1 Rewritten = true;
 2 while Rewritten do
 3 |   Rewritten = false;
 4 |   if IsEmpty(C) then return false;
 5 |   if IsTautology(C) then return true ;
 6 |   if ForwardSubsumption(C,N) then
 7 |   |   return true;
 8 |   end
 9 |   if ObviousReduction(C) then
10 |   |   Rewritten = true;
11 |   end
12 |   if SubtermContextualGroundRewriting(C,N) then
13 |   |   Rewritten = true;
14 |   end
15 end
16 return false;
```

The implementation is depicted in Algorithm 16 and uses tautology check, forward subsumption and obvious reductions from the reduction procedure of SPASS. These are the procedures implemented in SPASS except that they work with respect to the modified ordering procedures that interpret variables as constants. As explained above, the retrieval of candidate terms of forward subsumption remains unchanged.

Algorithm 16 expects as input a clause C and a clause set N. It reduces C with respect to N in the main loop. The reductions performed on the clause C in Algorithm 16 change C destructively. IsEmpty(C) checks whether the given clause is the empty clause. IsTautology(C) checks if $\models C$. This is realized via a congruence closure algorithm testing whether a positive literal is implied by the negative literals.

ForwardSubsumption(C, N) checks whether C is already subsumed by clauses from N.

$$\mathcal{R}\frac{\Gamma_1 \to \Delta_1 \qquad \Gamma_2 \to \Delta_2}{\Gamma_1 \to \Delta_1}$$

where $\Gamma_1 \subseteq \Gamma_2$ and $\Delta_1 \subseteq \Delta_2$.

ObviousReduction(C) consists of the following reduction rules

$$\mathcal{R}\frac{\Gamma \to \Delta, s \approx t, s \approx t}{\Gamma \to \Delta, s \approx t}$$

and

$$\mathcal{R}\frac{\Gamma, s \approx t, s \approx t \to \Delta}{\Gamma, s \approx t \to \Delta}$$

and

$$\mathcal{R}\frac{\Gamma \to \Delta, t \approx t}{\Gamma \to \Delta}$$

and

$$\mathcal{R}\frac{\Gamma, t \approx t \to \Delta}{\Gamma \to \Delta}$$

and

$$\mathcal{R}\frac{\Gamma, x \approx t \to \Delta}{\Gamma \to \Delta} \qquad \text{if} \quad x \notin (\Gamma \cup \Delta)$$

Further details can be found in the SPASS Handbook [WSK07].

12.3. Subterm Contextual Ground Rewriting

SubtermContextualGroundRewriting(C, N), depicted in Algorithm 17, implements subterm contextual ground rewriting. The variables occurring in C are interpreted as constants in the above explained sense. The call $general_{SDT}(N, u')$ (line 1) to the substitution tree index iteratively returns all generalizations s from N of u' together with the respective matcher σ. Then the procedure computes for each of the generalizations the literals and the clauses where they occur. The candidate clauses are then checked for the non-recursive side conditions of contextual rewriting. Because of the condition $vars(s) \supseteq vars(D)$ (line 5) the rewrite clause D is strongly universally reductive. This means that σ has all variables of D in its domain. The substitution σ replaces all variables of D by terms occurring in C. Therefore, $D\sigma$ contains only variables occurring in C which are assume to be constants. As a consequence, this procedure neither introduces any new Skolem constants nor does it change the precedence of them. Therefore, the ordering check (line 6) is implemented using the above explained, modified ordering modules treating variables as constants. Additionally,

Algorithm 17: SubtermContextualGroundRewriting

Input: clause $C[u[u'] \approx v]$, clause set N

1 **foreach** $(s, \sigma) \in general_{SDT}(N, u')$ **do**
2 $\quad Lits = \text{LiteralsContainingTerm}(s)$;
3 $\quad$ **foreach** $(s \approx t) \in Lits$ *s.t.* $s\sigma \succ t\sigma$ **do**
4 $\quad\quad D = \text{LiteralOwningClause}(s \approx t)$;
5 $\quad\quad$ **if** $vars(s) \supseteq vars(D) \wedge$
6 $\quad\quad\quad u[s\sigma] \approx v \succ s\sigma \approx t\sigma \wedge$
7 $\quad\quad\quad s\sigma$ *strictly maximal term in* $D\sigma \wedge$
8 $\quad\quad\quad \forall A \in Ante(D\sigma)$ $\text{GroundSubtermRedundant}(\Gamma \rightarrow A, N) \wedge$
9 $\quad\quad\quad \forall A \in Succ(D\sigma)$ $\text{GroundSubtermRedundant}(A \rightarrow \Delta, N)$
10 $\quad\quad$ **then**
11 $\quad\quad\quad$ **return** $C[u[t\sigma] \approx v]$;
12 $\quad\quad$ **end**
13 $\quad$ **end**
14 **end**

building the subproblems (line 8 – line 9) does also not change the Skolem constants nor introduce new Skolem constants. The two sets $Ante(D\sigma)$ and $Succ(D\sigma)$ denote the set of antecedent and succedent literals of $D\sigma$ without $s\sigma \approx t\sigma$, respectively. For each of these subproblems the procedure SubtermContextualGroundRewriting recursively calls the procedure GroundSubtermRedundant (Algorithm 16).

Interpreting variables as Skolem constants during the recursive application of GroundSubtermRedundant results exactly in the same behavior where explicitly new constant objects are introduced, but saves time and memory.

The implementation of subterm contextual rewriting (Definition 88) and subterm contextual literal elimination (Definition 89) is analogous to the implementation of subterm contextual ground rewriting. The difference is that the input clause C is not interpreted to be ground and the local side conditions (line 5 – line 7) are changed with respect to the definition of subterm contextual rewriting and subterm contextual literal elimination, respectively.

Algorithm 18 depicts the overall forward reduction procedure of SPASS where subterm contextual rewriting is integrated. Note that the input clause C is destructively changed during the reductions.

12.4. Integration of Unit and Non-Unit Rewriting

In addition to my first implementation [WW08] I have integrated unit and non-unit rewriting into subterm contextual rewriting. Considering the old Algorithm 18 standard rewriting (line 7), namely unit and non-unit rewriting, was implemented independently from subterm contextual rewriting (line 8). As a result, the previous implementation

Algorithm 18: ForwardReduction

Input: clause C, clause set N

```
1  Rewritten = true;
2  while Rewritten do
3  |  Rewritten = false;
4  |  if IsTautology(C) then return true;
5  |  if ObviousReduction(C) then Rewritten = true;
6  |  if ForwardSubsumption(C, N) then return true;
7  |  if Rewriting(C, N) then Rewritten = true;
8  |  if SubtermContextualRewrting(C, N) then Rewritten = true;
9  end
10 return (C);
```

searches the index structure for finding appropriate standard rewriting candidates and then searches the index again for finding candidates for subterm contextual rewriting. In order to save queries to the index and side condition checks, the new procedure (Algorithm 19) contains nested unit and non-unit rewriting.

Algorithm 19: SubtermContextualRewriting

Input: clause $C[u[u'] \approx v]$, clause set N

```
1  foreach (s, σ) ∈ general_SDT(N, u') do
2  |  Lits = LiteralsContainingTerm(s);
3  |  foreach (s ≈ t) ∈ Lits s.t. sσ ≻ tσ do
4  |  |  D = LiteralOwningClause(s ≈ t);
5  |  |  if IsUnit(C) ∧ IsUnit(Dσ) then
6  |  |  |  return C[u[tσ] ≈ v];
7  |  |  else if SubsumesBasic(C, Dσ) then
8  |  |  |  return  C[u[tσ] ≈ v];
9  |  |  else if u[sσ] ≈ v ≻ sσ ≈ tσ ∧
10 |  |  |  sσ strictly maximal term in Dσ ∧
11 |  |  |  ∀A ∈ Ante(Dσ) GroundSubtermRedundant(Γ → A, N) ∧
12 |  |  |  ∀A ∈ Succ(Dσ) GroundSubtermRedundant(A → Δ, N)
13 |  |  then
14 |  |  |  return C[u[tσ] ≈ v]
15 |  |
16 |  end
17 end
```

The procedure IsUnit (line 5) checks if the clause given as argument contains exactly one literal. If both C and $D\sigma$ are unit clauses then C can be rewritten. SubsumesBasic (line 7) checks if the literals of C except literal $u[u'] \approx v$ subsume all literals of $D\sigma$ except literal $s\sigma \approx t\sigma$. Analogously, SubtermContextualGroundRewriting (Algorithm 17) is extended by unit and non-unit rewriting.

The integration of unit and non-unit rewriting into subterm contextual rewriting potentially changes the proof search strategy because clauses are reduced in different order. Concerning Algorithm 18 rewriting (line 7) is performed on an input clause C before subterm contextual rewriting. The procedure implementing rewriting reduces the clause C using all clauses of N. If no further reduction with rewriting is possible then subterm contextual rewriting reduces C using N. After integrating standard rewriting into subterm contextual rewriting this is done in an interleaved way. The clause set N is processed only once. Each time a candidate clause is retrieved, the procedure checks if standard rewriting is possible. If it is not possible then it immediately checks whether subterm contextual rewriting is possible. This potentially changes the proof search strategy, because the clauses are reduced in a different order.

12.5. Fault Caching

Testing whether a term can be rewritten using subterm contextual rewriting might cause to perform many procedure calls because of the mutual recursive structure of the side conditions. To memorize terms that have been identified not to be reducible saves a lot of computation.

First, the new algorithm implementing subterm contextual ground rewriting, queries the cache each time a term is considered for rewriting. If the term is in the cache then this term is not considered for rewriting. If it is not in the cache and neither standard rewriting nor subterm contextual ground rewriting was possible then the algorithm inserts it into the cache. Once a term is inserted into the cache it remains there.

The cache operates globally in order to avoid as much computation as possible. This is an approximation because a term that is not reducible in the context of one clause might be reducible in the context of another clause. Further, for checking whether a term is in the fault cache, the implementation considers terms that are generalizations with respect to variable mappings. Because variables are interpreted as constants the cache might reject terms that are reducible with another ordering of the variables. Remember that variables are interpreted as Skolem constants. The fault cache is also compatible with splitting [Wei01]. If a term is inserted into the cache in a split branch that is not valid anymore, it does not produce wrong results because the cache is purely negative, i.e., it only excludes terms that could be possibly rewritten. As a consequence, this approach loses possible applications of a contextual rewriting step. Storing also terms that can be reduced would not work because if a term could be identified to be reducible in a split branch then this term does not have to be reducible in another branch. In this case backtracking updates of splitting have to consider cached clauses. Similarly, if a term is reducible in the context of one clause with contextual rewriting, this does not have to be the case in the context of another clause. However, the results in Chapter 13 show that this heuristic performs well on practical instances.

The cache itself is realized via a term index because this provides all the required functionality for storing and querying for terms. Furthermore, SPASS already provides

this data structure via substitution trees as explained above. Substitution trees return for a query term a generalization together with the respective substitution. Assume the check if a term t is already in the fault cache. If the substitution tree returns a term t' and a substitution σ then $t'\sigma = t$. If σ substitutes only variables of t' by variables then t is not subterm ground redundant if the same context is considered.

12.6. Context of Side Conditions

The current implementation of subterm contextual rewriting (Algorithm 19) does currently not consider the side condition as context; it only considers the clause set N. Consider the side conditions of subterm contextual rewriting which have the following form:

$$N_{C\tau}^{D\sigma\tau} \models (\Gamma \to \Delta)\tau. \tag{12.1}$$

This is equivalent to the following reasoning problem:

$$N_{C\tau}^{D\sigma\tau} \cup \{\Gamma\tau\} \models \Delta\tau. \tag{12.2}$$

Consequently, lines 11–12 of Algorithm 19 and lines 8–9 of Algorithm 17 could be adapted as follows, respectively.

$$\forall A \in \mathit{Ante}(D\sigma)\text{GroundSubtermRedundant}(\to A, N \cup \{\to \Gamma\}) \tag{12.3}$$

$$\forall A \in \mathit{Succ}(D\sigma)\text{GroundSubtermRedundant}(\to \Delta, N \cup \{\to A\}) \tag{12.4}$$

This is currently not implemented, because some additional investigation needs to be done in order to find efficient methods to maintain the insertion and deletion of the context which are feasible for application on practical problems.

13. Results

This chapter evaluates subterm contextual rewriting on two different applications. The first subsection 13.1 consecutively evaluates the three different variants of the implementation of subterm contextual rewriting on the TPTP. The second subsection depicts its application by continuing the introductory example.

13.1. Results on the TPTP

This section evaluates the implementations of subterm contextual rewriting in SPASS by comparing it to the current standard configuration of SPASS.

As test samples, I used the problems of the *TPTP 3.2.0* [SS98] which is a library consisting of 8984 problems for testing automated theorem proving systems. The reference run is SPASS version 3.1 that is version 3.0 extended by some bug fixes with default configuration. For the sample runs I have integrated subterm contextual rewriting and the respective improvements in this version of SPASS. All sample runs were performed with SPASS options set to *-RFRew=4 -RBRew=3 -RTaut=2*. This means that both subterm contextual rewriting and subterm contextual literal elimination are activated for forward rewriting, subterm contextual rewriting is activated for backward reductions and semantic tautology checks are activated. The hardware setup consisted of Opteron nodes running at speed of 2.4 GHz equipped with 4 GB RAM for each node. For the sample run as well as for the reference run the time limit was set to 300 seconds for each problem.

The problems of the TPTP are ranked from 0.00 to 1.00 indicating their level of difficulty. Basically, the value expresses how many of the current existing provers have been able to solve a particular problem. This means that problems with rating 1.00 have not been solved by any prover so far. For further details please consider [SS01]. The evaluation in this section compares SPASS containing the new improvements to the original SPASS with respect to the different rankings of problems. All proofs that SPASS with subterm contextual rewriting could find additionally were proof-checked.

First consider the implementation of subterm contextual rewriting and subterm contextual ground rewriting without integrated unit and non-unit rewriting and without fault caching. The results of running SPASS containing this implementation are depicted in Table 13.1. This version found 85 additional proofs and lost 143 proofs compared to the run without subterm contextual rewriting if considering all problems. If only problems with rank greater then 0.65 are considered then Table 13.1 shows that subterm

threshold	lost	won
0.00	143	85
0.50	80	69
0.65	48	61
0.80	3	42
0.90	0	21
1.00	0	5

Table 13.1.: Subterm Contextual rewriting, 300 seconds time limit

contextual rewriting solves more problems than it loses. The higher the rank the better are the results of subterm contextual rewriting compared to the reference version. If the threshold is greater than 0.9, SPASS with subterm contextual rewriting found 21 additional proofs and lost none. It even could solve five additional problems (problems with ranking 1.00).

At first, it was not clear why subterm contextual rewriting lost so many easy problems. Then by inspecting some of the lost problems two reasons could be identified. First, the actual proof found by the standard version of SPASS got lost through a subterm contextual rewriting application resulting in a different exploration of the search space. Second, a call to a contextual rewriting procedure took so long that SPASS did not finish within the time limit although the reference run terminated within milliseconds. Therefore, I improved the implementation of subterm contextual rewriting and subterm contextual ground rewriting, along the lines of Chapter 12, by integrating unit and non-unit rewriting into the procedure of subterm contextual rewriting and subterm contextual ground rewriting. This leads to the following section where the improved variant is evaluated.

13.1.1. Integrated Unit and Non-Unit Rewriting

The integration of unit and non-unit rewriting into the implementation of subterm contextual rewriting and subterm contextual ground rewriting improved the results significantly. Although more problems were lost than before more could be additionally solved. This improvement narrowed the difference between solved and lost problems from 58 to 33. Table 13.2 depicts the results.

Considering all problems the version with integrated unit and non-unit rewriting found 152 additional proofs and lost 119. It still solves hard problems whereas it is able to solve more easy ones.

threshold	lost	won
0.0	152	119
0.5	88	71
0.65	51	62
0.8	4	36
0.9	0	20
1.0	0	5

Table 13.2.: Integrated Unit and Non-Unit Rewriting, 300 seconds time limit

13.1.2. Fault Caching

After additionally integrating the fault cache in the subterm contextual ground rewriting procedure, the results further improved as Table 13.3 depicts. SPASS found 132 additional problems whereas it only lost 67. This implementation improved much on the easy problems but could also solve new difficult problems. Even one additional problem that has not been solved before (ranking 1.00).

threshold	lost	won
0.0	67	132
0.5	43	81
0.65	23	71
0.8	3	40
0.9	0	22
1.0	0	6

Table 13.3.: Subterm Contextual Rewriting with Caching, 300 seconds time limit

The following table compares SPASS with subterm contextual rewriting and all improvements with a time limit of 900 seconds to the reference run. As you can see almost half of the 67 lost problems could be regained by increasing the time limit.

The six new problems that SPASS with subterm contextual rewriting could solve are all from the software model checking category of the TPTP. The problems are: SWC308+1, SWC329+1, SWC345+1, SWC342+1, SWC261+1, SWC335+1. This is not a surprise as subterm contextual rewriting can for example employ conditional access function definitions for reduction. For example, a list element access function that first checks

threshold	lost	won
0.0	38	190
0.5	23	99
0.65	7	83
0.8	3	40
0.9	0	22
1.0	0	6

Table 13.4.: Subterm Contextual Rewriting with Caching, 900 sec run time

for emptiness is perfectly matched by the subterm contextual rewriting rule.

Although potentially deciding more satisfiable problems, the implementation of subterm contextual rewriting did not improve on satisfiable problems on the TPTP. The first implementation lost seven problems and did not terminate on problems in the time limit where the reference version did. The implementation integrating unit and non-unit rewriting even lost eight problems. But also for satisfiability problems it turned out that the fault cache is useful because the version containing the fault cache only lost one problem. Running the version with caching and a time limit of 900 seconds still lost this particular problem but additionally terminated on three problems. This is surprising because the example from Chapter 10 shows that subterm contextual rewriting is capable of increasing the number of problems on which SPASS can terminate. An explanation may be that the TPTP version 3.2.0 does not contain such problems.

13.2. Application to the Example from the Introduction

The following depicts the application of subterm contextual ground rewriting on the introductory example in detail. Therewith, the example shows that superposition together with the instance of contextual rewriting presented in this thesis, terminates on a problem on which the superposition calculus alone does not terminate; SPASS with subterm contextual rewriting is able to saturate the clause set from Chapter 10 whereas SPASS without subterm contextual rewriting is not. Recall that clause 10.9 could be reduced with clause 10.7 using contextual rewriting, in particular subterm contextual rewriting, if the side conditions are fulfilled. The ground clauses

$$\begin{aligned} i(x_1) \approx b, i(x_3) \approx b, i(x_2) \approx b, \\ q(y_1) \approx b, b \approx a \rightarrow i(x_1) \approx b \end{aligned} \tag{13.1}$$

$$\begin{aligned} i(x_1) \approx b, i(x_3) \approx b, i(x_2) \approx b, q(y_1) \approx b, \\ b \approx a \rightarrow q(f(x_2, r(x_3, y_1))) \approx b \end{aligned} \tag{13.2}$$

must be entailed by clauses from N_C. Clause 8 is a tautology whereas clause 9 can be rewritten with clause 3 to

$$i(x_1) \approx b, i(x_3) \approx b, i(x_2) \approx b, \\ q(y_1) \approx b, a \approx b \to a \approx b \tag{13.3}$$

using subterm contextual ground rewriting if in addition the clauses

$$i(x_1) \approx b, i(x_3) \approx b, i(x_2) \approx b, \\ q(y_1) \approx b, a \approx b \to i(x_2) \approx b \tag{13.4}$$

$$i(x_1) \approx b, i(x_3) \approx b, i(x_2) \approx b, q(y_1) \approx b, \\ a \approx b \to q(r(x_3, y_1)))) \approx b \tag{13.5}$$

are also entailed by clauses from N_C. Clause 13.4 is a syntactic tautology and clause 13.5 is subsumed by clause 10.6.

14. Computing in Minimal Models

If all clauses of a given clause set N are universally reductive, then it is decidable whether a ground equation is valid in the minimal model N_I of N [GS93]. This chapter proves that subterm contextual rewriting is a decision procedure for ground clauses in this context.

First, the minimal model N_I of N with respect to set inclusion is defined. A rewrite relation R is defined from a ground clause set N by induction over the clause ordering $\succ$ as follows. Suppose $E_{C'}$ and $R_{C'}$ have been defined for a ground clause C with $C \succ C'$. Then

$$R_C = \bigcup_{C \succ C'} E_{C'}$$

Further, if $C = \Gamma \rightarrow \Delta, s \approx t$ of a clause in N such that (i) C is reductive for $s \approx t$ (ii) s is irreducible by R_C, (iii) $\Gamma \subset R_C^*$ and (iv)$\Delta \cap R_C^* = \emptyset$ then

$$E_C = \{s \approx t\}$$

otherwise $E_C = \emptyset$. R_C^* is the transitive, reflexive closure of R_C. The rewrite system R is defined as $R = \bigcup_C E_C$.

For a non-ground clause set N, N_I is the above model with respect to all ground instances of all clauses from N.

Lemma 91. *[GS93] Let N be a saturated, finite and universally reductive set of clauses. Then, it is decidable whether a ground equation $s \approx t$ is valid in the minimal model N_I.*

Following [GS93], if N is a saturated, finite and universally reductive set of clauses and $s \approx t$ is ground, then deciding whether $s \approx t$ is valid in N_I is equivalent to deciding $s \approx t \in R^*$. More precisely, if N is a saturated, finite and universally reductive set of clauses then the construction of R provides a minimal model N_I for N. For deciding whether $s \approx t$ is valid in N, s and t have to be rewritten using R to their normal forms and can then be checked for equality.

The sequel shows that whenever a rewriting step on $s \approx t$ is possible with respect to R then subterm contextual rewriting with respect to the clause set N can also perform this step.

In order to apply a subterm contextual rewriting step, the ordering restrictions have to be fulfilled. Therefore, the equation $s \approx t$ can equivalently be transformed into the

clause $top \approx top \rightarrow s \approx t$ such that top is the greatest symbol in the signature. This is also used in [GS93] for applying general contextual rewriting as a decision procedure for $s \approx t$. Additionally, condition 2 of Definition 87 has to be changed to

$$u[s\sigma] \approx v \succ s\sigma \approx t\sigma \text{ or } top \approx top \in C'$$

This is an approximation of condition 2 of general contextual rewriting like condition 2 of Definition 87. Consequently, adapting condition 2 does not change completeness.

Lemma 92. *Let N be a saturated, finite and universally reductive set of clauses. Subterm contextual rewriting can decide whether a ground equation $s \approx t$ is valid in the minimal model N_I.*

Proof:
First, equivalently transform $s \approx t$ to the clause $C = \ top \approx top \rightarrow s \approx t$. By Lemma 91 $s \approx t$ is decidable using R. If there is a relation $l \approx r \in R$ that can rewrite s or t, then there is a ground instance $D = \Gamma \rightarrow \Delta,\ l \approx r$ of a clause in N producing $l \approx r$, i.e. D is reductive. In order to perform an actual subterm contextual rewriting step on C using D, the side conditions have to be fulfilled.

1. *D is universally reductive for $l \approx r$ and, therefore, $l\sigma$ is ground and $\succ$ can be decided.*
2. *The adapted condition holds because $top \approx top \in C$.*
3. *Condition 3 requires that $top \approx top \rightarrow s' \approx t'$ is ground subterm redundant for $s' \approx t'$ in Γ and clauses from N_C. This is a strictly smaller problem (with respect to $\succ$) than $s \approx t$ and therefore, together with an analogous lemma for subterm contextual ground rewriting, follows inductively.*
4. *Condition 4 requires that $s' \approx t' \rightarrow$ is ground subterm redundant for $s' \approx t'$ in Δ and clauses from N_C. This is a strictly smaller problem (with respect to $\succ$) than $s \approx t$ and therefore, together with an analogous lemma for subterm contextual ground rewriting, follows inductively.*

Due to the previous lemma, the validity of ground literals in minimal models can be decided with subterm contextual rewriting. As a consequence, subterm contextual rewriting can also decide the validity of ground clauses in minimal models.

15. Conclusion

In summary, contextual rewriting is costly but it helps solving difficult problems. It actually gains more problems than it loses and is able to solve many difficult problems. In addition, it is a decision procedure for ground equations in the minimal model of a universally reductive equational theory. The implementation described in this thesis and used for the experiments is integrated into SPASS version 3.5 and can be obtained from the SPASS homepage (`http://spass-prover.org/`).

Part III.

Summary

16. Summary

The complexity of a set of first-order formulas results from its size and from the complexity of the problem described by its formulas. In this thesis, I have advanced superposition based automated theorem proving by accomplishing two goals. First, I have developed decision procedures for ontologies that consist of several million formulas. These procedures decide the satisfiability of ontologies and answer complex queries containing alternating quantifiers with respect to minimal model semantics. Second, I have developed a powerful reduction rule for complex, particularly undecidable, first-order reasoning problems. This new reduction rule enables superposition to solve more complex problems, and it extends the number of problems for which superposition is a decision procedure.

Decision Procedures for Ontologies

In this thesis, I have presented the first superposition based reasoning procedure for efficiently deciding the satisfiability of ontologies that consist of several million axioms from BSH-Y2. The set BSH-Y2 is a subset of the Bernays–Schönfinkel Horn fragment with equality. It is able to represent the YAGO ontology as well as large parts of the ontologies SUMO (SUMO-Y2) and CYC (CYC-Y2). In general, verifying the satisfiability in the Bernays–Schönfinkel Horn fragment is EXPTIME complete.

Additionally, I have developed the first efficient, sound, and complete reasoning procedure that decides the entailment of complex formulas, which contain arbitrary quantifier alternations, with respect to minimal model semantics in ontologies with several million formulas. Because minimal model reasoning is beyond standard first-order reasoning, I have developed a sophisticated query answering procedure based on a finite quantifier elimination algorithm.

The index, which stores and manages the formulas occurring in a reasoning problem, is the central data structure for successful automated theorem proving. This is because it is accessed several thousand times during one reasoning loop. Consequently, in order to successfully reason about ontologies consisting of several million clauses, sophisticated index data structures are necessary. For this reason, I have developed a new index called filtered context tree. The filtered context tree invents a filtering technique that discriminates the symbols occurring in an ontology. The filtering excludes whole parts of the context tree from the search space of a retrieval operation. As a result, the filtering provides efficient implementations of the retrieval operations for context trees. The invention of this filtering technique was pivotal for successfully reason about ontologies because SPASS, without the new index, was already unable to load the YAGO ontology.

I have implemented the new procedures together with the new indexing in SPASS. The resulting version is called SPASS–Y2. It decides the consistency of YAGO within 16 minutes, the consistency of SUMO-Y2 in 53 minutes, and it finds inconsistencies of CYC-Y2 within one minute. The evaluation in this work shows that SPASS-Y2 is currently the only system that can decide the satisfiability of all three ontologies. As a consequence, SPASS-Y2 is the first tool for efficiently deciding the satisfiability of ontologies that consist of several million clauses.

SPASS-Y2 is also the first superposition based query answering tool that formally, completely, and efficiently answers complex first-order queries containing arbitrary quantifier alternations. This procedure answers queries with respect to minimal model semantics, and the experiments have shown that it answers queries within a few seconds. As a consequence, SPASS-Y2 is a practically useful query answering tool.

Reductions for General Automated Theorem Proving

In order to obtain efficient reasoning procedures for complex, particularly undecidable, first-order problems, sophisticated reductions that keep the search space small are essential. For this reason, I have developed subterm contextual rewriting. Subterm contextual rewriting is a decidable instance of contextual rewriting which is undecidable, in general.

With the help of subterm contextual rewriting, many difficult problems that could not be solved before, can be solved now. Furthermore, it extends the number of problems for which superposition is a decision procedure. I have also shown that subterm contextual rewriting is a decision procedure for ground equations in the minimal model of a universally reductive equational theory. Subterm contextual rewriting is the first effective decision procedure for this problem.

Bibliography

[ABK+07] Soeren Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. DBpedia: a nucleus for a web of open data. In *Proceedings of the 6th international The semantic web and 2nd Asian conference on Asian semantic web conference*, ISWC'07/ASWC'07, pages 722–735, Busan, Korea, 2007. Springer-Verlag. ACM ID: 1785216.

[Bar03] C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.

[BBVF10] Georg Boenn, Martin Brain, Marina De Vos, and John Fitch. Automatic music composition using answer set programming. *CoRR*, abs/1006.4948, 2010.

[BCM+03] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, March 2003.

[Bec04] Dave Beckett. RDF/XML syntax specification (Revised). http://www.w3.org/TR/REC-rdf-syntax/, 2004.

[BG90] Leo Bachmair and Harald Ganzinger. On restrictions of ordered paramodulation with simplification. In *CADE*, volume 449 of *Lecture Notes in Computer Science*, pages 427–441. Springer, 1990.

[BG94] Leo Bachmair and Harald Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *Journal of Logic and Computation*, 4(3):217–247, 1994.

[BG98] Leo Bachmair and Harald Ganzinger. Ordered chaining calculi for first-order theories of transitive relations. *J. ACM*, 45(6):1007–1049, 1998.

[BG01] Leo Bachmair and Harald Ganzinger. Resolution theorem proving. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, pages 19–99. Elsevier and MIT Press, 2001.

[BGW93] L. Bachmair, H. Ganzinger, and U. Waldmann. Superposition with simplification as a decision procedure for the monadic class with equality. In *Computational Logic and Proof Theory*, volume 713 of *LNCS*, pages 83–96, 1993.

[BR95] Adel Bouhoula and Michaël Rusinowitch. Implicit induction in conditional theories. *J. Autom. Reasoning*, 14(2):189–235, 1995.

[CGL09] Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. A general datalog-based framework for tractable query answering over ontologies. In *Proceedings of the twenty-eighth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '09, pages 77–86, New York, NY, USA, 2009. ACM.

[CGT89] S. Ceri, G. Gottlob, and L. Tanca. What you always wanted to know about datalog (and never dared to ask). *IEEE Trans. on Knowl. and Data Eng.*, 1:146–166, March 1989.

[Cla78] K. L Clark. Negation as failure. *Logic and data bases*, pages 293—322, 1978.

[Der82] Nachum Dershowitz. Orderings for term-rewriting systems. *Theor. Comput. Sci.*, 17:279–301, 1982.

[DFK+07] Julian Dolby, Achille Fokoue, Aditya Kalyanpur, Aaron Kershenbaum, Edith Schonberg, Kavitha Srinivas, and Li Ma. Scalable semantic retrieval through summarization and refinement. In *AAAI*, pages 299–304, 2007.

[DLP94] Didier Dubois, Jérôme Lang, and Henri Prade. Automated reasoning using possibilistic logic: Semantics, belief revision, and variable certainty weights. *IEEE Trans. Knowl. Data Eng.*, 6(1):64–71, 1994.

[ECD+04] Oren Etzioni, Michael Cafarella, Doug Downey, Stanley Kok, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld, and Alexander Yates. Web-scale information extraction in knowitall. In *Proceedings of the 13th conference on World Wide Web - WWW '04*, page 100, New York, NY, USA, 2004.

[Fel98] Christiane Fellbaum. *WordNet: an electronic lexical database*. MIT Press, 1998.

[FGHP08] Ulrich Furbach, Ingo Glöckner, Hermann Helbig, and Björn Pelzer. Loganswer - a deduction-based question answering system (system description). In *Proceedings of the 4th international joint conference on Automated Reasoning*, IJCAR '08, pages 139–146, Berlin, Heidelberg, 2008. Springer-Verlag.

[FLHT01] Christian G. Fermüller, Alexander Leitsch, Ullrich Hustadt, and Tanel Tamet. Resolution decision procedures. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume II, chapter 25, pages 1791–1849. Elsevier, 2001.

[Gan87] Harald Ganzinger. A completion procedure for conditional equations. In Stéphane Kaplan and Jean-Pierre Jouannaud, editors, *CTRS*, volume 308 of *LNCS*, pages 62–83. Springer, 1987.

[GdN99] Harald Ganzinger and Hans de Nivelle. A superposition decision procedure for the guarded fragment with equality. In Giuseppe Longo, editor, *Proceedings of the 14th Annual IEEE Symposium on Logic in Computer Science (LICS-99)*, pages 295–303, Trento, Italy, 1999. IEEE Computer Society, IEEE.

[GKK+11] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, M. Schneider, and S. Ziller. A portfolio solver for answer set programming: Preliminary report. In *Lecture Notes in Artificial Intelligence*, volume 6645, pages 352–357. Springer-Verlag, 2011.

[GKKS11a] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. Challenges in answer set solving. In *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning: Essays in Honor of Michael Gelfond*, volume 6565, pages 74–90. springer, 2011.

[GKKS11b] Martin Gebser, Roland Kaminski, Arne König, and Torsten Schaub. Advances in *gringo* series 3. In James P. Delgrande and Wolfgang Faber, editors, *Logic Programming and Nonmonotonic Reasoning - 11th International Conference, LPNMR 2011, Vancouver, Canada, May 16-19, 2011. Proceedings*, volume 6645 of *Lecture Notes in Computer Science*, pages 345–351. Springer, 2011.

[GL90] Michael Gelfond and Vladimir Lifschitz. Logic programs with classical negation. pages 579–597. MIT Press, Cambridge, MA, USA, 1990.

[GMW97] Harald Ganzinger, Christoph Meyer, and Christoph Weidenbach. Soft typing for ordered resolution. In William McCune, editor, *Automated Deduction – CADE-14*, volume 1249 of *Lecture Notes in Computer Science*, pages 321–335. Springer Berlin / Heidelberg, 1997.

[GN94] Harald Ganzinger and Robert Nieuwenhuis. The saturate system 1994. http://www.mpi-sb.mpg.de/SATURATE/Saturate.html, 1994.

[GNN01] Harald Ganzinger, Robert Nieuwenhuis, and Pilar Nivela. Context trees. In Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *IJCAR*, volume 2083 of *Lecture Notes in Computer Science*, pages 242–256. Springer, 2001.

[GNN04] Harald Ganzinger, Robert Nieuwenhuis, and Pilar Nivela. Fast term indexing with coded context trees. *J. Autom. Reasoning*, 32(2):103–120, 2004.

[Gra96] Peter Graf. *Term Indexing*, volume 1053 of *Lecture Notes in Computer Science*. Springer, 1996.

[Gru95] Thomas R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *Int. J. Hum.-Comput. Stud.*, 43:907–928, December 1995.

[GS93] Harald Ganzinger and Jürgen Stuber. Inductive theorem proving by consistency for first-order clauses. In *The Third International Workshop on Conditional Term Rewriting Systems, Extended Abstracts*, pages 130–135. Teubner Verlag, 1993.

[GST07] Martin Gebser, Torsten Schaub, and Sven Thiele. Gringo : A new grounder for answer set programming. In Chitta Baral, Gerhard Brewka, and John S. Schlipf, editors, *Logic Programming and Nonmonotonic Reasoning, 9th International Conference, LPNMR 2007, Tempe, AZ, USA, May 15-17, 2007, Proceedings*, volume 4483 of *Lecture Notes in Computer Science*, pages 266–271. Springer, 2007.

[GSTV11] Martin Gebser, Torsten Schaub, Sven Thiele, and Philippe Veber. Detecting inconsistencies in large biological networks with answer set programming. *TPLP*, 11(2–3):323–360, 2011.

[HM01] Volker Haarslev and Ralf Möller. Racer system description. In *Proceedings of the First International Joint Conference on Automated Reasoning*, IJCAR '01, pages 701–706, London, UK, UK, 2001. Springer-Verlag.

[HMS07] Ullrich Hustadt, Boris Motik, and Ulrike Sattler. Reasoning in Description Logics by a Reduction to Disjunctive Datalog. *Journal of Automated Reasoning*, 39(3):351–384, 2007.

[HSB+11] Johannes Hoffart, Fabian M. Suchanek, Klaus Berberich, Edwin Lewis-Kelham, Gerard de Melo, and Gerhard Weikum. Yago2: Exploring and querying world knowledge in time, space, context, and many languages. In Sadagopan Srinivasan, Krithi Ramamritham, Arun Kumar, M. P. Ravindra, Elisa Bertino, and Ravi Kumar, editors, *Proceedings of the 20th international conference companion on World Wide Web (WWW 2011)*, pages 229–232, Hyderabad, India, 2011. Association for Computing Machinery (ACM), ACM.

[HSG04] Ullrich Hustadt, Renate A. Schmidt, and Lilia Georgieva. A survey of decidable first-order fragments and description logics. *Journal of Relational Methods in Computer Science*, 1:251–276, 2004.

[HV06] Ian Horrocks and Andrei Voronkov. Reasoning support for expressive ontology languages using a theorem prover. In *Foundations of Information and Knowledge Systems*, volume 3861, pages 201–218. Springer Berlin / Heidelberg, 2006.

[HV11] Krystof Hoder and Andrei Voronkov. Sine qua non for large theory reasoning. In Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors, *Automated Deduction – CADE-23*, volume 6803 of *Lecture Notes in Computer Science*, pages 299–314. Springer Berlin / Heidelberg, 2011.

[HW08] Matthias Horbach and Christoph Weidenbach. Superposition for fixed domains. In *CSL*, volume 5213 of *Lecture Notes in Computer Science*, pages 293–307, Bertinoro, Italy, 2008. Springer.

[HW10] Matthias Horbach and Christoph Weidenbach. Superposition for fixed domains. *ACM Trans. Comput. Log.*, 11(4), 2010.

[IMB+09] Harold Ishebabi, Philipp Mahr, Christophe Bobda, Martin Gebser, and Torsten Schaub. Application of asp for automatic synthesis of flexible multiprocessor systems from parallel programs. In Esra Erdem, Fangzhen Lin, and Torsten Schaub, editors, *LPNMR*, volume 5753 of *Lecture Notes in Computer Science*, pages 598–603. Springer, 2009.

[JMW98] F. Jacquemard, C. Meyer, and C. Weidenbach. Unification in extensions of shallow equational theories. In *Proceedings of RTA-98*, volume 1379 of *LNCS*, pages 76–90. Springer, 1998.

[KB70] D. E. Knuth and P. B. Bendix. Simple word problems in universal algebras. In I. Leech, editor, *Computational problems in Abstract Algebras*, pages 263–297. Pergamon Press, 1970.

[KKS11] Yevgeny Kazakov, Markus Krötzsch, and František Simancík. Concurrent classification of el ontologies. In *Proceedings of the 10th international conference on The semantic web - Volume Part I*, ISWC'11, pages 305–320, Berlin, Heidelberg, 2011. Springer-Verlag.

[Kor08] K. Korovin. iProver – an instantiation-based theorem prover for first-order logic (system description). In A. Armando, P. Baumgartner, and G. Dowek, editors, *Proceedings of the 4th International Joint Conference on Automated Reasoning, (IJCAR 2008)*, volume 5195 of *Lecture Notes in Computer Science*, pages 292–298. Springer, 2008.

[KW11] Evgeny Kruglov and Christoph Weidenbach. Sup(t) decides first-order logic fragment over ground theories. In *MACIS 2011: Fourth International Conference on Mathematical Aspects of Computer and Information Sciences*, 2011.

[Len95] Douglas B. Lenat. Cyc: a large-scale investment in knowledge infrastructure. *Commun. ACM*, 38(11):33–38, November 1995.

[LPF+06] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The DLV system for knowledge representation and reasoning. *ACM Trans. Comput. Log.*, 7(3):499–562, 2006.

[LTW09] Carsten Lutz, David Toman, and Frank Wolter. Conjunctive query answering in the description logic EL using a relational database system. In *IJCAI*, pages 2070–2075, 2009.

[McC03] William McCune. Otter 3.3 reference manual. *CoRR*, cs.SC/0310056, 2003.

[Min82] Jack Minker. On indefinite databases and the closed world assumption. In D. Loveland, editor, *6th Conference on Automated Deduction*, volume 138 of *Lecture Notes in Computer Science*, pages 292–308. Springer Berlin / Heidelberg, 1982. 10.1007/BFb0000066.

[MS06] Boris Motik and Ulrike Sattler. A comparison of reasoning techniques for querying large description logic aboxes. In Miki Hermann and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning*, volume 4246 of *Lecture Notes in Computer Science*, pages 227–241. Springer Berlin / Heidelberg, 2006.

[MW97] William McCune and Larry Wos. Otter. *Journal of Automated Reasoning*, 18(2):211–220, 1997.

[NBG+00] M. Nogueira, M. Balduccini, M. Gelfond, R. Watson, and M. Barry. An a-prolog decision support system for the space shuttle. In *In PADL 2001*, pages 169–183. Springer, 2000.

[NHRV01] Robert Nieuwenhuis, Thomas Hillenbrand, Alexandre Riazanov, and Andrei Voronkov. On the evaluation of indexing techniques for theorem proving. In Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *Automated Reasoning, First International Joint Conference, IJCAR 2001, Siena, Italy, June 18-23, 2001, Proceedings*, volume 2083 of *LNCS*, pages 257–271. Springer, 2001.

[NN93] Pilar Nivela and Robert Nieuwenhuis. Saturation of first-order (constrained) clauses with the *Saturate* system. In Claude Kirchner, editor, *Rewriting Techniques and Applications, 5th International Conference, RTA-93*, volume 690 of *Lecture Notes in Computer Science, LNCS*, pages 436–440, Montreal, Canada, June 16–18, 1993. Springer.

[NP01a] Ian Niles and Adam Pease. Towards a standard upper ontology. In *Proceedings of the international conference on Formal Ontology in Information Systems - Volume 2001*, FOIS '01, pages 2–9, Ogunquit, Maine, USA, 2001. ACM. ACM ID: 505170.

[NP01b] Ian Niles and Adam Pease. Towards a standard upper ontology. In *FOIS '01*, pages 2–9, Ogunquit, Maine, USA, 2001. ACM.

[NR01] Robert Nieuwenhuis and Albert Rubio. Paramodulation-based theorem proving. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, pages 371–443. Elsevier and MIT Press, 2001.

[NW08] Thomas Neumann and Gerhard Weikum. Rdf-3x: a risc-style engine for rdf. *PVLDB*, 1(1):647–659, 2008.

[OL80] Ross Overbeek and Ewing Lusk. Data structures and control architecture for implementation of theorem-proving programs. In Wolfgang Bibel and Robert Kowalski, editors, *5th Conference on Automated Deduction Les*

Arcs, France, July 8–11, 1980, volume 87 of *Lecture Notes in Computer Science*, pages 232–249. Springer Berlin / Heidelberg, 1980.

[OWL09] W3c owl working group: Owl 2 web ontology language: Document overview, 2009. W3C Recommendation (October 27, 2009).

[PHP08] A. Philpot, E.H. Hovy, and P. Pante. The omega ontology. In *Ontology and the Lexicon*. Cambridge University Press, 2008.

[Pla84] David A. Plaisted. Complete problems in the first-order predicate calculus. *J. Comput. Syst. Sci.*, 29(1):8–35, 1984.

[PS07] Adam Pease and Geoff Sutcliffe. First order reasoning on a large ontology. In Geoff Sutcliffe, Josef Urban, and Stephan Schulz, editors, *ESARLT*, volume 257 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

[PS08] Simone Paolo Ponzetto and Michael Strube. WikiTaxonomy: a large scale knowledge resource. In *ECAI*, pages 751–752, 2008.

[PSST10] Adam Pease, Geoff Sutcliffe, Nick Siegel, and Steven Trac. Large theory reasoning with sumo at casc. *AI Commun.*, 23:137–144, April 2010.

[PW07] Björn Pelzer and Christoph Wernhard. System description: E– KRHyper. In Frank Pfenning, editor, *Automated Deduction – CADE–21*, volume 4603 of *Lecture Notes in Computer Science*, pages 508–513. Springer Berlin / Heidelberg, 2007.

[Rei77a] Raymond Reiter. Deductive Question-Answering on relational data bases. In *Logic and Data Bases*, pages 149–177, 1977.

[Rei77b] Raymond Reiter. On closed world data bases. In *Logic and Data Bases*, pages 55–76, 1977.

[RR00] Jun Rao and Kenneth A. Ross. Making b^+-trees cache conscious in main memory. In Weidong Chen, Jeffrey F. Naughton, and Philip A. Bernstein, editors, *SIGMOD Conference*, pages 475–486. ACM, 2000.

[RRG05] Deepak Ramachandran, Pace Reagan, and Keith Goolsbey. First-orderized researchcyc: Expressivity and efficiency in a common-sense ontology. In *In Papers from the AAAI Workshop on Contexts and Ontologies: Theory, Practice and Applications*, 2005.

[RSV01] I. V. Ramakrishnan, R. C. Sekar, and Andrei Voronkov. Term indexing. In *Handbook of Automated Reasoning*, pages 1853–1964. 2001.

[RV01] Alexandre Riazanov and Andrei Voronkov. Vampire 1.1. In Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *Automated Reasoning*, volume 2083, pages 376–380–380–376–380–380. Springer Berlin / Heidelberg, 2001.

[SB05] Stephan Schulz and Maria Paola Bonacina. On Handling Distinct Objects in the Superposition Calculus. In B. Konev and S. Schulz, editors, *Proc. of the 5th International Workshop on the Implementation of Logics, Montevideo, Uruguay*, pages 66–77, 2005.

[Sch99] Renate A. Schmidt. Decidability by resolution for propositional modal logics. *Journal of Automated Reasoning*, 22:379–396, 1999. 10.1023/A:1006043519663.

[Sch02] Stephan Schulz. E - a brainiac theorem prover. *Ai Communications*, 15(2-3):111–126, 2002.

[Sch12] Anne Schlicht. *Scaling Up Description Logic Reasoning by Distributed Resolution*. PhD thesis, Universität Mannheim, 2012.

[SKW07] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: A Core of Semantic Knowledge. In *16th international World Wide Web conference (WWW 2007)*, pages 697–706, New York, NY, USA, 2007. ACM Press.

[SKW08] F Suchanek, G Kasneci, and G Weikum. YAGO: a large ontology from wikipedia and WordNet. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(3):203–217, September 2008.

[SN98] Timo Soininen and Ilkka Niemelä. Developing a declarative rule language for applications in product configuration. In Gopal Gupta, editor, *Practical Aspects of Declarative Languages*, volume 1551 of *Lecture Notes in Computer Science*, pages 305–319. Springer Berlin / Heidelberg, 1998.

[SPG+07] E. Sirin, B. Parsia, B. C Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical owl-dl reasoner. *Web Semantics: science, services and agents on the World Wide Web*, 5(2):51–53–51–53, 2007.

[SPT11] Robert Simmons, Frank Pfenning, and Bernardo Toninho. Distributed deductive databases, declaratively: The l10 logic programming language. In *ACM SIGPLAN 2011 X10 Workshop*, June 2011.

[SS89] Manfred Schmidt-Schauß. *Computational Aspects of an Order-Sorted Logic with Term Declarations*, volume 395 of *Lecture Notes in Computer Science*. Springer, 1989.

[SS98] G. Sutcliffe and C.B. Suttner. The TPTP Problem Library: CNF Release v1.2.1. *Journal of Automated Reasoning*, 21(2):177–203, 1998.

[SS01] G. Sutcliffe and C.B. Suttner. Evaluating General Purpose Automated Theorem Proving Systems. *Artificial Intelligence*, 131(1-2):39–54, 2001.

[SS03] Andreas M. Seidl and Thomas Sturm. Boolean quantification in a first-order context. In V. G. Ganzha, E. W. Mayr, and E. V. Vorozhtsov, editors, *Computer Algebra in Scientific Computing. Proceedings of the CASC 2003*, pages 329–345. Institut für Informatik, Technische Universität München, München, Germany, 2003.

[SS11] Michael Schneider and Geoff Sutcliffe. Reasoning in the owl 2 full ontology language using first-order automated theorem proving. In Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors, *Automated Deduction – CADE-23*, volume 6803 of *Lecture Notes in Computer Science*, pages 461–475. Springer Berlin / Heidelberg, 2011.

[SSW+09] Martin Suda, Geoff Sutcliffe, Patrick Wischnewski, Manuel Lamotte-Schubert, and Gerard de Melo. External sources of axioms in automated theorem proving. In *KI*, pages 281–288, 2009.

[Sta09] S. Staab. *Handbook on ontologies*. Springer Verlag, 2009.

[Sut10] Geoff Sutcliffe. The tptp world - infrastructure for automated reasoning. pages 1–12, 2010.

[Sut11] Geoff Sutcliffe. The 5th ijcar automated theorem proving system competition - casc-j5. *AI Commun.*, 24(1):75–89, 2011.

[SWW10] Martin Suda, Christoph Weidenbach, and Patrick Wischnewski. On the Saturation of YAGO. In Jürgen Giesl and Reiner Hähnle, editors, *Automated Reasoning*, volume 6173 of *Lecture Notes in Computer Science*, pages 441–456. Springer Berlin / Heidelberg, 2010.

[Syr98] Tommi Syrjänen. Implementation of local grounding for logic programs with stable model semantics. Technical Report B18, Helsinki University of Technology, Digital Systems Laboratory, Espoo, Finland, October 1998.

[TH06] D. Tsarkov and I. Horrocks. Fact++ description logic reasoner: System description. *Automated Reasoning*, pages 292–297–292–297, 2006.

[VHLPs08] F. Van Harmelen, V. Lifschitz, B. Porter, and ScienceDirect (Online service). *Handbook of knowledge representation*. Elsevier, 2008.

[WBH+02] Christoph Weidenbach, Uwe Brahm, Thomas Hillenbrand, Enno Keen, Christian Theobald, and Dalibor Topic. SPASS version 2.0. In Andrei Voronkov, editor, *Proceedings of the 18th International Conference on Automated Deduction (CADE-18)*, volume 2392 of *Lecture Notes in Artificial Intelligence*, pages 275–279, Kopenhagen, Denmark, 2002. Springer.

[WDF+09] Christoph Weidenbach, Dilyana Dimova, Arnaud Fietzke, Rohit Kumar, Martin Suda, and Patrick Wischnewski. Spass version 3.5. In *Automated Deduction - CADE-22*, volume 5663, pages 140–145. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[Wei98] Christoph Weidenbach. *Sorted Unification and Tree Automata*, chapter 9, pages 291–320. Applied Logic. Kluwer, Dordrecht, The Netherlands, January 1998.

[Wei01] Christoph Weidenbach. Combining superposition, sorts and splitting. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume 2, chapter 27, pages 1965–2012. Elsevier, 2001.

[WSK07] Christoph Weidenbach, Renate Schmidt, and Enno Keen. Spass handbook version 3.0. Contained in the documentation of SPASS Version 3.0, 2007.

[WW08] Christoph Weidenbach and Patrick Wischnewski. Contextual rewriting in spass. In Boris Konev, Renate A. Schmidt, and Stephan Schulz, editors, *PAAR/ESHOL*, volume 373 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[WW10] Christoph Weidenbach and Patrick Wischnewski. Subterm contextual rewriting. *AI Commun.*, 23:97–109, April 2010.

[Zha93] Hantao Zhang. Implementing contextual rewriting. In *CTRS '92: Proceedings of the Third International Workshop on Conditional Term Rewriting Systems*, pages 363–377, London, UK, 1993. Springer-Verlag.

Index

T

U

V

Y