Satisfiability Checking and Query Answering for Large Ontologies

Christoph Weidenbach[1] and Patrick Wischnewski[12]

[1] Max Planck Institute for Informatics, Saarbrücken, Germany
[2] Universität des Saarlandes, Saarbrücken, Germany
{weidenbach,wischnew}@mpi-inf.mpg.de

Abstract

In this paper we develop a sound, complete and terminating superposition calculus plus a query answering calculus for the BSH-Y2 fragment of the Bernays – Schönfinkel Horn class of first-order logic. BSH-Y2 can be used to represent expressive ontologies. In addition to checking consistency, our calculus supports query answering for queries with arbitrary quantifier alternations. Experiments on BSH-Y2 (fragments) of several large ontologies show that our approach advances the state of the art.

1 Introduction

In addition to research in description logics [1], reasoning in ontologies has recently drawn a lot of attention in automated theorem proving [4, 19, 16, 7] as well as database theory [9, 3]. The approaches differ in the expressiveness of the considered logics, the supported reasoning tasks as well as the quality of the reasoning procedures. A focus in description logics is on the sound and complete computation of the concept hierarchy, whereas theorem proving and data base approaches typically consider existentially quantified queries. Concerning the theorem proving approaches we can distinguish complete methods from incomplete ones. Whereas the former guarantee completeness and consistency of the ontology [19], the latter consider very expressive ontologies [4, 16, 7] and aim at providing useful query answering.

The approach of this paper is to keep completeness plus consistency checking, but to push the border of expressivity. Previously [19] we have shown effective satisfiability testing for the language *BSH-Y1* consisting of clauses of the form

$\to P(a_1,\ldots,a_n)$	Ground Fact	$R(x,y),R(y,z)\to R(x,z)$	Transitivity
$S(x)\to T(x)$	Subsort Relation	$R(x,y),R(x,z)\to y\approx z$	Functionality

where all function symbols are constants enjoying the unique name assumption. The language covers the YAGO ontology. We have shown that we can decide satisfiability of the YAGO ontology consisting of 10m clauses over 2m constants of the above form in about one hour by superposition based saturation. Existentially quantified queries with respect to the saturated YAGO ontology can then typically be answered in the range of seconds.

In this paper we consider an extended language, called *BSH-Y2*, where in addition to the above clauses we consider clauses of the form

$P_1(t_{11},\ldots,t_{1n_1}),\ldots,P_k(t_{k1},\ldots,t_{kn_k})\to$	Negative Clauses
$P_1(t_{11},\ldots,t_{1n_1}),\ldots,P_k(t_{k1},\ldots,t_{kn_k})\to P(s_1,\ldots,s_m)$	Defined Relations

where the t_{ij}, s_j are either constants or variables and all variables of the s_j show up in some t_{ij} (range restriction). Due to a further developed superposition calculus (Section 3) and its implementation, the extended BSH-Y2 language is still suitable to decide large ontologies. We tested our implementation on three large ontologies: the extension YAGO++ of the YAGO ontology fully covered by BSH-Y2 (10m clauses), the BSH-Y2 fragment of the SUMO ontology as it appears in the TPTP library [12], serving about 90% of the TPTP SUMO version (82k clauses, SUMO-Y2), and on the CYC ontology as it appears in the TPTP library [13], serving about 30% of the TPTP CYC version (1m clauses, CYC-Y2). We considered the SUMO ontology and the CYC ontology from the CASC–23 competetion [20]. SPASS-Y2 can check consistency for the YAGO++ and SUMO-Y2 ontologies, where we found 2 logical inconsistencies in SUMO-Y2. So far we have not been able to check consistency of CYC-Y2. We stopped after finding and debugging 35 logical inconsistencies. For further details, see Section 5.

Furthermore, we provide complete reasoning support for queries with arbitrary quantifier alternations, introduced in Section 4. Again queries are typically answered in the range of seconds with respect to the minimal model of a saturated ontology via a special query answering calculus. Note that then, completeness turns into soundness for queries with quantifier alternations. For example, answering a query of the form $\exists x\ \forall y\ \Phi$ requires complete reasoning for the universal quantifier in order to obtain a sound result for the overall query. In Section 5, we provide experimental data for query answering with respect to YAGO++ and SUMO-Y2. The paper ends with a short summary and further discussion of related work. The proofs of the theorems are available in a technical report [23].

2 Preliminaries

In this paper we follow the notations from [21]. We assume a first-order language over a signature Σ as usual. We use x, y, z to denote variables, a, b and c to denote constants, s, t, l, r to denote terms P, Q, S to denote predicate symbols, A, B to denote atoms C, D to denote clauses and N to denote a set of clauses. Let vars be the function returning all variables of a term, an atom, and a clause, respectively. We write σ for a substitution.

We only consider Horn clauses which we write in implication form $\Gamma \rightarrow \Delta$ with Γ is a multiset of literals and Δ is either the empty set or a singleton set containing one atom. We call a positive ground unit clause $(\rightarrow A)$ a fact and a negative ground unit clause $(A \rightarrow)$ a negative fact. For the empty clause we write $\square$.

An inference is a rule of the form

$$\frac{C_1 \quad \dots \quad C_n}{D}$$

where the clause D can be derived from the premises $C_1, \dots, C_n$. We say an inference is ground iff all clauses $C_1, \dots, C_n$ and D are ground. An inference system is a collection of inference rules. As usual for the superposition calculus, inferences will be restricted to maximal positive literals and negative literals that are either maximal or selected.

We say that a ground clause C is *redundant* with respect to a set of clauses N if there exists a set $\{C_1, \dots, C_k\}$ of ground instances of clauses from N such that C is true in every model of $\{C_1, \dots, C_k\}$ and $C \succ C_j$, for all j with $1 \leq j \leq k$. A non-ground clause is called *redundant* if all its ground instances are.

A ground inference π is *redundant* with respect to N if either one of its premises is redundant in N, or else there exists a set $\{C_1, \dots, C_k\}$ of ground instances of clauses from N such that the conclusion of π is true in every model of $\{C_1, \dots, C_k\}$ and $C \succ C_j$, for all j with $1 \leq j \leq k$, where C is the maximal premise of π. A non-ground inference is called *redundant* if all its ground instances are redundant.

We say that a set of clauses N is *saturated up to redundancy* with respect to some inference system, if all inferences from N are redundant.

A reduction rule reduces the search space by deleting clauses or by reducing clauses to simpler ones. A reduction is denoted as

$$\mathcal{R}\,\frac{C_1 \quad \dots \quad C_n}{\begin{array}{c} D_1 \\ \vdots \\ D_m \end{array}}$$

where the clause above the bar $C_1, \dots, C_n$ are replaced by the clauses below the bar $D_1, \dots, D_m$. A reduction rule implements a special redundancy criteria.

A *Herbrand* interpretation I is a set of ground atoms. Each ground atom A is called true in I if $A \in I$. It is called false in I if $A \notin I$. A negated atom $\neg A$ is true in I if $A \notin I$. A ground clause C is called true in I if one of its literals is true in I. We write $I \models C$ in this case.

2.1 Admissible Term Ordering

For reasoning in large domain problems efficiently handling transitivity is important due to the fact that the standard superposition approach is too prolific in this context; it computes the whole transitive closure. The chaining calculus [2] has been designed for efficiently dealing with transitivity in general by avoiding the computation of the transitive closure in many cases. We have integrated the chaining calculus into our new reasoning calculus.

The chaining calculus is defined in terms of an extension of the usual reduction ordering on terms. This extension is called admissible and defined as follows.

An ordering $\succ$ on ground terms and literals is called *admissible* if

- it is well-founded and total on ground terms and literals,
- it is *compatible with reduction on maximal subterms*, i.e. $L \succ L'$ 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.
  - $\neg A \succ A$ 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.

For implementing an actual admissible ordering a triple (max_L, p_L, min_L) is associated with each literal L. Two literals are compared by lexicographically comparing their associated triples. For the comparison of the first and last component of the triples the superposition term ordering $\succ$ is used and for comparing the middle component the ordering $1 > 0$ is used. The individual members of the triples are defined as follows: If L is of the form $Q(s,t)$ for a transitive predicate

Q we set max_L to the maximum of s and t, and min_L to the minimum of the two terms (with respect to $\succ$). If L is of the form A or $\neg A$ for some atom A the top symbol of which is not a transitive predicate, we set $max_L = A$ and $min_L = \top$, where $\top$ is special symbol minimal in the term ordering $\succ$. We set $p_L = 1$, if L is negative, and 0 otherwise.

2.2 Minimal Model

The chaining calculus assumes a given clause set N which does not contain any transitivity axioms. Instead, it assumes that the respective predicates are marked as transitive. These predicates are treated specially by the chaining rules. The candidate model for a set of Horn clauses that entails the transitive closure is defined in terms of rewrite proofs. The rewrite proof from the term l to r via the transitive predicate Q, is denoted as $l \Downarrow_Q^{R_C} r$. A detailed introduction to transitive rewrite proofs can be found in [2].

The following defines a minimal candidate model for a set N of Horn clauses which is also a model of the transitive closure of N. Assume that N does not contain any transitivity axioms and assume that the respective transitive predicates are marked as transitive.

Definition (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.

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

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.

3 Superposition for BSH-Y2

In this section we define the BSH-Y2 class and present our new superposition calculus for BSH-Y2.

3.1 The BSH-Y2 Class

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). It is defined below.

We call a clause $D = \Gamma \rightarrow P(t_1, \ldots, t_n)$ a *definition* for the predicate P if it is range restricted, i.e. vars$(P(t_1, \ldots, t_n)) \subseteq$ vars(Γ). We also say that P is *defined* by D. A predicate

Q is called *P–dependent* in a clause set N if Q is defined by a clause $D = \Gamma \to Q(t_1, \ldots, t_n)$ in N and (i) $P(s'_1, \ldots, s'_{n'}) \in \Gamma$ or (ii) there is a predicate R with $R(s''_1, \ldots, s''_{n''}) \in \Gamma$ that is P-dependent. If Q is not P–dependent we call it *P–independent.* A definition $D = \Gamma \to P(t_1, \ldots, t_n)$ is *acyclic* in a clause set N if P is P–independent. A predicate Q is called *transitive dependent* in N iff (i) Q is transitive or (ii) there is a definition $C \in N$ with $C = \Gamma \to Q(t_1, \ldots, t_n)$ and there is a transitive dependent predicate P with $P(s_1, \ldots, s_n) \in \Gamma$. Otherwise, we call Q *transitive independent.*

The *BSH-Y2* class consists of the following types of BSH clauses:

$\to P(a, b)$	facts
$R(x, y), R(x, z) \to y \approx z$	functionality axioms
$R(x, y), R(y, z) \to R(x, z)$	transitivity axioms
$P_1(t_{11}, \ldots, t_{1n_1}), \ldots, P_k(t_{k1}, \ldots, t_{kn_k}) \to$	negative clauses
$P_1(t_{11}, \ldots, t_{1n_1}), \ldots, P_k(t_{k1}, \ldots, t_{kn_k}) \to P(s_1, \ldots, s_m)$	acyclic definitions
$S_1(x) \to S_2(x)$	subsort relations

where the subset of subsort relations is acyclic and we further assume the unique name assumption for the BSH-Y2 meaning that different constants represent different domain elements.

3.2 Calculus for BSH-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 Section 5 confirm this. In [19], we have developed a sound and complete calculus which uses hyperresolution together with the chaining calculus. 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:

$$\text{bornIn}(x, y), \text{locatedIn}(y, z) \to \text{bornInTr}(x, z) \tag{1}$$

where locatedIn is transitive.

Therefore, we have developed a new calculus for BSH-Y2 that performs a two layered-reasoning. It separates reasoning about non-transitive predicates from reasoning about transitive predicates via dedicated inference rules. These rules are depicted in the following.

The calculus is defined with respect to a clause set N containing clauses from BSH-Y2. Let $\mathcal{S}_N$ be the sort theory contained in N. We switch from the simple clause notation introduced in Section 2 to a clause notation $\Theta \parallel \Gamma \to \Delta$ where Θ contains solely the sort atoms (monadic atoms) interpreted as negated sort atoms. This notation helps in defining the below rules as it explicitly separates sort atoms from others. During the saturation the sort atoms are treated independently from all other atoms via the rules *Empty Sort* , *Sort Simplification*, and *Static Soft Typing.* Actually, this simulates a particular ordering and selection strategy for these atoms on the standard calculus [5]. More precisely, $T \succ S$ for all subsort declarations $S(x) \to T(x)$. This ordering is well-defined because there are no cycles in $\mathcal{S}_N$. Whenever a clause has an unsolved constraint, this constraint is selected.

An unsolved constraint Θ of a clause $\Theta \parallel \Gamma \to \Delta$ either contains an atom $T(x)$ such that $x \notin \text{vars}(\Gamma \cup \Delta)$ or an atom $T(a)$ for some constant a. Finally, all sort predicates S occurring in N are smaller than any other predicate occurring in N. The fact that the monadic predicates are treated separately allows more efficient implementations for their calculus rules.

The sort theory $\mathcal{S}_N$ is static [5] meaning that $N_I \models S(a)$ iff $\mathcal{S}_N \models S(a)$ and this property is invariant on the saturation of N while fixing $\mathcal{S}_N$ from the beginning. This is due to the fact that all positive sort atoms in N occur either as facts or as subsort declarations. Hence, when deriving a clause $S(a), \Theta \,\|\, \Gamma \rightarrow \Delta$ with $\mathcal{S}_N \not\models S(a)$, the clause is a tautology and can be deleted. This is exploited by the implementation of our new calculus. Note also that the relations $\mathcal{S}_N \models S(a)$ and $\mathcal{S}_N \models \exists x\, S_1(x) \wedge \ldots \wedge S_n(x)$ can be efficiently decided by specific algorithms [21].

The chaining calculus [2] assumes that all transitivity axioms are deleted from the clause set N and the respective atoms are marked as transitive. We assume the set Tr that contains all transitive predicate symbols of N.

The rule *OECut* [18] ensures that the minimal model N_I respects the unique name assumption, namely $N_I \models a \not\approx b$ for two different constants a and b occurring in N. So the disequations are not explicitly added to the clause set N.

The superposition calculus for the BSH-Y2 is the following set of inference and reduction rules.

Non-Transitive Reasoning

Ordered Hyperresolution for BSH-Y2 (HyperY2)

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

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 $i \in \{1, \ldots n\}$, respectively, and $A_i\sigma$ are strictly maximal in $(\Theta_i \,\|\, \Gamma_i \rightarrow A_i)\sigma$.

Object Equality Cutting (OECut)

$$\frac{\|\, \rightarrow a \approx b}{\square},$$

where a and b are two different constants.

Transitive Reasoning

Ordered Chaining for BSH-Y2 (OChainY2)

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

where $Q \in \text{Tr}$ 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 $(\Theta \,\|\, \Gamma \rightarrow Q(t, r))\sigma$, $l\sigma \not\succeq s\sigma$, $r\sigma \not\succeq t\sigma$, and there are only transitive literals in Γ.

Negative Chaining for BSH-Y2 (NChainY2)

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

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 $(\Theta \,\|\, \Gamma, Q(t,r) \rightarrow)\sigma$, and there are only transitive literals in Γ.

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

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 $(\Theta \,\|\, \Gamma, Q(t,r) \rightarrow)\sigma$, and there are only transitive literals in Γ.

Ordered Resolution for BSH-Y2 (OReY2)

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

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 $(\Theta \,\|\, \Gamma, Q(s_1,s_2) \rightarrow)\sigma$, and there are only transitive literals in Γ.

Sort Reasoning

Empty Sort

$$\frac{S(x), \Theta \,\|\, \Gamma \rightarrow \Delta}{(\Theta \,\|\, \Gamma \rightarrow \Delta)\sigma},$$

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

Sort Simplification

$$\mathcal{R}\frac{S(a), \Theta \,\|\, \Gamma \rightarrow \Delta}{\Theta \,\|\, \Gamma \rightarrow \Delta} \;,$$

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

Static Soft Typing

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

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

Theorem (Decison Procedure for BSH-Y2)**.** *The BSH-Y2 calculus is sound, complete and terminating for BSH-Y2.*

3.3 Implementation

For successfully saturating a clause set from BSH-Y2, an efficient implementation of the new calculus rules is essential. In our implementation we avoid to generate clauses that are redundant and can, therefore, immediately be removed from the search space.

In particular, every time an application of hyperresolution derives a clause $\Theta, S(a) \parallel \Gamma \rightarrow A$ an application of sort simplification becomes possible on the ground sort instance $S(a)$. If $\mathcal{S} \models S(a)$ the clause is reduced to $\Theta \parallel \Gamma \rightarrow A$. If $\mathcal{S} \not\models S(a)$ then the clause is a tautology and can be deleted. Therefore, we have integrated this sort reasoning in the implementation of hyperresolution. This means that the clause $\Theta \parallel \Gamma \rightarrow A$ where Θ does not contain any further ground sort instances is derived.

Additionally, our implementation of the calculus relies on the efficient term indexing data structure *Filtered context trees* that we have introduced in [19]. We have integrated the implementation of our new calculus together with the data structures in the theorem prover SPASS [22]. We call the resulting version SPASS-Y2.

4 Query Answering

In this section we present a query language with arbitrarily many quantifier alternations and the corresponding sound and complete query answering procedure. This procedure answers queries with respect to the minimal model of a clause set from BSH-Y2.

4.1 Query Language

Consider the language Φ defined in terms of the below syntax

$$\Phi \quad := \quad \Gamma \quad | \quad \forall x(\Gamma \rightarrow \Phi) \quad | \quad \exists x(\Gamma \wedge \Phi) \quad | \quad \top$$

where Γ is a conjunction of atoms.

In order to guarantee completeness of our new query answering procedure, we require further restrictions on the query language. We call a formula of the language Φ *variable shielded* iff it is either ground or it is of the form $\exists x(\Gamma \wedge \Phi')$ or $\forall x(\Gamma \rightarrow \Phi')$ 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.

We call a formula φ a *query* if it is a variable shielded sentence from the language Φ. Further, we assume that a variable is bound by at most one quantifier in a query.

Note, that shielding of the variables is not a real restriction because it can always be achieved via a special predicate entity, s.t. for every constant c of the signature entity(c) is entailed by the minimal model of the respective ontology.

4.2 Query Answering Calculus

Let N be the saturation of a set of clauses from BSH-Y2 with respect to the superposition calculus of Section 3 and $\square \notin N$. Further, let $\mathcal{S}_N$ be the sort theory contained in N. Our query answering calculus is composed of deterministic rule system with respect to N and $\mathcal{S}_N$ and consists of three calculus rules; one for each type of query: *existential query*, *universal query* and *ground query.* The efficiency of our rule system is based on the following observation.

Lemma. *Let N be the saturation of a clause set from the BSH-Y2 with $\Box \notin N$ and let A be a transitive independent ground atom. Then $N_I \models A$ iff there is a ground substitution σ and a clause $\Theta \parallel \to B \in N$ with $B\sigma = A$ and $\mathcal{S}_N \models \Theta\sigma$.*

Note, all transitive independent definitions are ground instantiated during the saturation. 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 our calculus, is again a query because all variables of Φ are shielded. 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. Because of the previous lemma, it is sufficient to consider only clauses of the form $\Theta \parallel \to A$ from N as the right premisses.

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-sorted check which is quasi-linear [15]. The entailment check in condition 4 is performed by exhaustively applying the saturation calculus of Section 3 with a set of support strategy. Note, our new calculus is a decision procedure for this minimal model reasoning problem because of Theorem 3.2 and [8].

Existential query

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

if there is a grounding substitution σ such that

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

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

if there is a grounding substitution σ such that

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

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

if there is a grounding substitution σ such that

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

4.3 Query Answering Procedure

If N is the saturation of a clause set from BSH-Y2 in terms of the calculus of Section 3 then Algorithm 1 implements the query answering procedure which is sound and complete with respect to the minimal model N_I. The strategy corresponds to a quantifier elimination over finite domains.

Algorithm 1: AnswerQuery

```
Input: Query $\Phi$, saturated clause set $N$
if $\Phi = \top$ then return true
else if $\Phi = \exists\overline{x}.\Gamma \wedge \Phi'$ then
    foreach $\Phi'\sigma \in \text{ext}(\Phi, \text{N})$ do
        if AnswerQuery($\Phi'\sigma$,$N$) then return true;
    end
    return false;
else if $\Phi = \forall\overline{x}.\Gamma \to \Phi'$ then
    foreach $\Phi'\sigma \in \text{unv}(\Phi, N)$ do
        if $\neg$AnswerQuery($\Phi'\sigma, N$) then return false;
    end
    return true;
else if $\text{gnd}(\Phi, N) = \text{true}$ then return true
else return false
```

The algorithm expects as its input a query Φ and the clause set N. First, the algorithm checks whether the given query Φ is an existential quantified, a universally quantified or a ground query. Then it computes the set of all subqueries obtained by applying the respective calculus rule. The set $\text{ext}(\Phi, N)$ is the set of all subqueries from applying the rule *Existential query* to Φ and N. Likewise, the set $\text{unv}(\Phi, N)$ is the set of all subqueries from applying the rule *Universal query* to Φ and N. Finally, $\text{gnd}(\Phi, N)$ is the result of the application of *Ground query*. If $\text{gnd}(\Phi, N)$ is true then the algorithm returns true otherwise it returns false. Our algorithm processes a query from the outer query to the inner subquery. Checking if $N_I \models \forall x(\Gamma \to \Phi')$ requires to check if each ground instance of Γ is also contained in the set of instances of Φ'. Since, we have to compute the ground instances of Γ anyway, we process the queries from the outer to the inner query.

Our implementation of the query answering calculus follows exactly Algorithm 1. The rules are implemented following the implementation of hyperresolution style rules. One exception to the straight forward implementation is the condition 4 that checks if a transitive dependent atom A is entailed by N_I. We do not use the whole reasoning engine of Spass-Y2 for this purpose. Instead, we have implemented a procedure that simulates several derivation steps in hyperresolution style macro steps while keeping an efficient implicit representation of the query clause.

Theorem. *Let Φ be a query, N the saturation of a clause set from the BSH-Y2 with respect to the saturation calculus of Section 3. If N_I is the minimal model of N, then*

$$N_I \models \Phi \Leftrightarrow \text{AnswerQuery}(\Phi, N) = \text{true}$$

5 Experiments

We have extended the automated theorem prover Spass 3.8 [22] with the saturation and query answering calculus presented in this paper. We call this new version Spass-Y2. Spass-Y2 is sound, complete and terminating for BSH-Y2 and additionally provides a query answering engine for queries with quantifier alternations.

For our experiments we used the SUMO and CYC ontologies from the CASC–23 [20] competition. In order to obtain the clause set SUMO-Y2, we extracted all clauses belonging to the BSH-Y2 language from the SUMO ontology file CSR003+2.ax of the TPTP. The clause set SUMO-Y2 contains $82,064$ which is about 90% of CSR003+2.ax. The remaining 10% cannot be expressed in the BSH-Y2 fragment. In particular, the relations s__instance and s__subclass can be expressed in BSH-Y2. Likewise, for CYC-Y2 we extracted the BSH-Y2 clauses from the base knowledge of the CYC TPTP file CSR002+5.ax. The clause set CYC-Y2 contains about $1,033,447$ clauses out of $3,341,996$. We consider only the BSH-Y2 fragment of the base knowledge of CYC in CYC-Y2 because this has already a high level of inconsistency. In other words, we do not consider the microtheories of CYC for our experiments. The YAGO++ ontology that we used for our experiments includes the first-order representation of the YAGO ontology plus further axioms. For example, we added the following definition which is an refinement of the relation locatedIn: $\text{locatedIn}(x, y) \rightarrow \text{locatedInTr}(x, y)$, removed the transitivity axiom for locatedIn and added a transitivity axiom for locatedInTr. This allows us to check the relation locatedIn for additional properties like functionality and antisymmetry. The relation locatedInTr together with the respective transitivity axioms represents the transitive closure of the original locatedIn relation. The resulting clause set YAGO++ contains $9,918,724$ clauses over $2m$ constants.

We ran our experiments on a 2 x Intel Xeon Processor X5660 (12 MB Cache, 2.80 GHz) Debian Linux machine with 96 GB RAM with Spass-Y2 compiled as 64 bit binary. We require a 64 bit architecture for our experiments because Spass-Y2 needs to address around 20 GB RAM for the saturation of the YAGO++ ontology.

5.1 Saturation Procedure

In our experiments we compare clasp 2.0.4 with the grounder gringo [6], DLV [11], Vampire 1.8 [14], E 1.4 [17], iProver 0.8.1 [10] and Spass 3.8 [22] with Spass-Y2. Spass 3.8 contains already some of our data structure from our previous work [19] but not an implementation of the calculi presented here. All provers were called using the recommended default settings with a time limit of 100 min. We ran each of these tools with YAGO++, SUMO-Y2 and CYC-Y2.

	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	1 min		kbs	25 min		unsat	26 sec
E		kbs	6 min		t.o.	100 min		kbs	3 min
iProver		kbs	1 min	$967,678$	t.o.	100 min		kbs	8 sec
Spass3.8	$49,848,842$	sat	60 min	$1,530,025$	t.o.	100 min	$18,907,803$	t.o.	100 min
Spass-Y2	$2,722,246$	sat	16min	$790,691$	sat	45min	328,904	unsat	1 min

Figure 1: Evaluation of Spass-Y2

The results are depicted in Figure 1. The first column shows the tool and the second 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), depicted in the result column. We have also tested the model finders FIMO 0.2, E-Darwin 1.4, and Paradox 0.4, but none of these tools could find a model for any of the three ontologies within 100 minutes. Except for Spass-Y2, none of these tools could saturate all three ontologies and find inconsistencies. clasp performed nicely on SUMO-Y2 and CYC-Y2, but it could not find a model of YAGO++ because it run out of main memory (96 GB).

5.2 Query Answering Procedure

We have tested the query answering abilities of Spass-Y2 in the standard first-order semantics as well as in minimal model semantics. For the evaluation in terms of the standard first-order semantics, we have tested the 20 queries of the SUMO category of the CASC-23 competition. Before answering the queries we have saturated the SUMO-Y2 ontology and removed the logical inconsistencies. We have identified two logical inconsistencies of the SUMO ontology as used in CASC-23. Then we applied the saturation procedure of Spass-Y2 with a, in this case, complete set of support strategy in order to find a proof for the respective query. 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 conjectures of the remaining two problems cannot be formulated in the BSH-Y2 language. Spass-Y2 could have answered all of these questions in terms of the inconsistent SUMO ontology (principle of explosion). After identifying and fixing 35 inconsistencies in the base knowledge of CYC, we did not consider CYC for further experiments because it is questionable what an answer in terms of an inconsistent ontology means.

We have tested the query answering procedure of Section 4 of Spass-Y2 by running the procedure on the following queries with respect to the saturated clause set of the YAGO++ ontology.

Each of the following queries regard a particular feature of our query language or the BSH-Y2 language. This includes quantifier alternations, transitive dependent and transitive independent definitions.

$$
\begin{aligned}
Q_1 &= \exists x(\mathrm{politician}(x) \wedge \mathrm{physicist}(x))\\
Q_2 &= \exists x, y, z(\mathrm{hasSuccessor}(x, \mathrm{GeorgeWBush}) \wedge \mathrm{graduatedFrom}(x, z) \wedge\\
&\qquad \mathrm{graduatedFrom}(y, z) \wedge \mathrm{isMarriedTo}(x, y))\\
Q_3 &= \exists x, y(\mathrm{bornIn}(\mathrm{Angela\_Merkel}, y) \wedge \mathrm{locatedIn}(x, y) \wedge \mathrm{country}(y))\\
Q_4 &= \exists x, y(\mathrm{bornIn}(x, y) \wedge \forall z.\, \mathrm{hasChild}(x, z) \rightarrow \mathrm{bornIn}(z, y))\\
Q_5 &= \exists x(\mathrm{bornIn}(x, y) \wedge \mathrm{politician}(x) \wedge \mathrm{locatedIn}(x, \mathrm{Europe}) \wedge \mathrm{physicist}(x))\\
Q_6 &= \exists x(\mathrm{bornIn}(x, \mathrm{Hamburg}) \wedge \mathrm{politician}(x) \wedge \mathrm{physicist}(x) \wedge\\
&\qquad \mathrm{hasSuccessor}(\mathrm{Helmut\_Schmidt}, x))\\
Q_7 &= \forall x(\mathrm{politicianOf}(x, \mathrm{Germany}) \rightarrow \exists y, z.\, \mathrm{hasSuccessor}(y, x) \wedge \mathrm{bornIn}(y, z) \wedge\\
&\qquad \mathrm{locatedIn}(z, \mathrm{Germany}))\\
Q_8 &= \exists x(\mathrm{politician}(x) \wedge \mathrm{bornInCountry}(x, \mathrm{Germany}))
\end{aligned}
$$

The time that SPASS-Y2 needed to answer this queries are depicted in the below table.

Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8
0:00.79	0:01.13	1:10.28	0:00.33	0:09.20	0:00.00	0:03.36	0:29.95

The automated theorem proving systems that participated in the CASC-23 LTB division are not suitable in order to answer these queries. They are incomplete because of their axiom selection strategy. In the case of a quantifier alternation, completeness is required for soundness. Furthermore, these systems do not provide a minimal model query answering procedure.

SPASS-Y2 answers most of the queries in a few seconds with respect to minimal model semantics. Our implementation returns "Yes" or a counter example for universal queries and "No" or a complete set of answers for existential queries. For example the query Q_6 returns $\{x \mapsto \mathrm{AngelaMerkel}\}$. 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/`.

6 Conclusion

We have presented a sound and complete superposition calculus for BSH-Y2 covering YAGO++ and large portions of SUMO and CYC. The implementation SPASS-Y2 can effectively decide satisfiability for all three ontologies, where all other systems we have tested fail on at least one input set. clasp performed nicely on SUMO-Y2 and CYC-Y2 but failed on YAGO++ due to the 2m constants and transitive relations preventing efficient grounding. Our results on SUMO-Y2 show that winning the respective CASC competition category can be easily done by focusing on one of the logical inconsistencies. Our results on CYC show, where we stopped after finding and debugging 35 inconsistencies, that it is highly inconsistent. So keeping completeness, but further developing theory and implementation in order to be able to effectively check consistency for large problems can lead to useful insights.

In addition, we provide a new calculus for query processing supporting queries with arbitrary quantifier alternations with respect to consistent and saturated clause sets of YAGO++ and SUMO-Y2. Typical query response times for complex queries are in the range of seconds. There is currently no other implementation of an automated reasoning procedure that supports queries with quantifier alternations with respect to large ontologies out of the BSH-Y2 fragment.

References

[1] 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.

[2] Leo Bachmair and Harald Ganzinger. Ordered chaining calculi for first-order theories of transitive relations. *J. ACM*, 45(6):1007–1049, November 1998.

[3] 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. ACM, 2009.

[4] Ulrich Furbach, Ingo Glöckner, Hermann Helbig, and Björn Pelzer. Loganswer - a deduction-based question answering system (system description). In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *IJCAR*, volume 5195 of *LNCS*, pages 139–146. Springer, 2008.

[5] Harald Ganzinger, Christoph Meyer, and Christoph Weidenbach. Soft typing for ordered resolution. In William McCune, editor, *CADE 14*, volume 1249 of *LNCS*, pages 321–335. Springer, 1997.

[6] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, M. Schneider, and S. Ziller. A portfolio solver for answer set programming: Preliminary report. In *LNAI*, volume 6645, pages 352–357. Springer, 2011.

[7] Krytof Hoder and Andrei Voronkov. Sine qua non for large theory reasoning. In Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors, *CADE-23*, volume 6803 of *LNCS*, pages 299–314. Springer, 2011.

[8] Matthias Horbach and Christoph Weidenbach. Superposition for fixed domains. *ACM Trans. Comput. Log.*, 11(4), 2010.

[9] 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.

[10] K. Korovin. iProver – an instantiation-based theorem prover for first-order logic (system description). In A. Armando, P. Baumgartner, and G. Dowek, editors, *IJCAR*, volume 5195 of *LNCS*, pages 292–298. Springer, 2008.

[11] 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.

[12] 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.

[13] 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.

[14] Alexandre Riazanov and Andrei Voronkov. Vampire 1.1. In Rajeev Gor, Alexander Leitsch, and Tobias Nipkow, editors, *IJCAR*, volume 2083, pages 376–380–380–376–380–380. Springer, 2001.

[15] Manfred Schmidt-Schauß. *Computational Aspects of an Order-Sorted Logic with Term Declarations*, volume 395 of *LNCS*. Springer, 1989.

[16] 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, *CADE-23*, volume 6803 of *LNCS*, pages 461–475. Springer, 2011.

[17] Stephan Schulz. E - a brainiac theorem prover. *Ai Communications*, 15(2-3):111–126, 2002.

[18] 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.

[19] Martin Suda, Christoph Weidenbach, and Patrick Wischnewski. On the Saturation of YAGO. In Jrgen Giesl and Reiner Hähnle, editors, *IJCAR*, volume 6173 of *LNCS*, pages 441–456. Springer, 2010. An extended version of this article can be found in the Technical Report MPI-I-2010-RG1-001.

[20] Geoff Sutcliffe. The cade-23 automated theorem proving system competition - casc-23. *AI Commun.*, 25(1):49–63, 2012.

[21] 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.

[22] Christoph Weidenbach, Dilyana Dimova, Arnaud Fietzke, Rohit Kumar, Martin Suda, and Patrick Wischnewski. Spass version 3.5. In *CADE-22*, volume 5663, pages 140–145. Springer, 2009.

[23] Christoph Weidenbach and Patrick Wischnewski. Satisfiability checking and query answering for large ontologies. Technical Report MPI-I-2011-RG1-001, Max Planck Institute for Informatics, 2011.