Reasoning Support for Expressive Ontology Languages Using a Theorem Prover

Ian Horrocks and Andrei Voronkov [⋆]

The University of Manchester
{horrocks|voronkov}@cs.man.ac.uk

Abstract. It is claimed in [45] that first-order theorem provers are not efficient for reasoning with ontologies based on description logics compared to specialised description logic reasoners. However, the development of more expressive ontology languages requires the use of theorem provers able to reason with full first-order logic and even its extensions. So far, theorem provers have extensively been used for running experiments over TPTP containing mainly problems with relatively small axiomatisations. A question arises whether such theorem provers can be used to reason in real time with large axiomatisations used in expressive ontologies such as SUMO. In this paper we answer this question affirmatively by showing that a carefully engineered theorem prover can answer queries to ontologies having over 15,000 first-order axioms with equality. Ontologies used in our experiments are based on the language KIF, whose expressive power goes far beyond the description logic based languages currently used in the Semantic Web.

State-of-the-art theorem provers for first-order logic (FOL) are highly sophisticated and efficient programs. Moreover, they are very flexible tools and can be tuned to a number of applications. For example, VAMPIRE [35] provides a large collection of parameters that can be used to give better performance for various classes of applications. In addition, VAMPIRE implements a number of literal selection functions and internally contains a library for defining such functions in a simple way; this makes it possible to simulate various proof-search algorithms and even provide decision procedures for decidable classes of first-order logic (see, e.g., [23]).

However, there was a common belief that provers like VAMPIRE cannot directly be used for efficient reasoning with very large ontologies using expressive languages such as KIF [14] for two reasons. Firstly, these provers are optimised for reasoning with relatively small axiomatisations. Secondly, they do not support some extensions of FOL required in KIF.

In this paper we describe an adaptation of VAMPIRE to support reasoning for expressive ontology languages and present experimental results which show that it can be used for efficient reasoning with large ontologies using extensions of the first-order language.

⋆ The authors are partially supported by grants from EPSRC.

This paper is structured as follows. In Section 1 we briefly overview expressive languages for ontologies, including the language KIF, and FO provers. In Section 2 we describe the adaptation of VAMPIRE for reasoning with large ontologies. For our experiments we selected ontologies implemented in KIF since they offer a high degree of sophistication compared to ontologies using Description Logic (DL) based languages, and there are publicly available KIF-based ontologies containing thousands of FO formulas with equality. However, our adaptation is quite general, and could be used for other expressive ontology languages.

Note that there is no way to compare VAMPIRE with description logic provers on these ontologies, since their subsets corresponding to description logic languages and hardly interesting and not representative.

Section 3 contains a summary and a description of future work. Finally, in the appendix we demonstrate the efficiency and advanced features of VAMPIRE by showing two (out of a large number of) inconsistency proofs found by it in SUMO and a terrorism ontology.

1 Introduction

Expressive Languages for Ontologies Ontologies play a major role in the Semantic Web where they are widely used in, e.g., bio-informatics, medical terminologies and other knowledge management applications [5, 44, 47, 41, 34, 40]. They are also of increasing importance in the Grid, where they may be used, e.g., to support semantic based discovery, execution and monitoring of Grid services [9, 46, 11].

State of the art ontology languages, such as DAML+OIL [48] and OWL [3], are based on expressive description logics (DLs). This establishes a firm formal foundation for the language, e.g., by providing well-defined semantics and a broad understanding of the computational properties of key inference problems; it also allows applications to exploit the reasoning services provided by highly optimised DL reasoners such as FaCT and Racer [19, 15, 13]. It is widely recognised, however, that the expressive power of such languages is inadequate in some applications, and in particular applications related to the discovery and composition of Web and Grid services. This has led to efforts to develop languages based on more expressive logics up to and including full first-order predicate logic [17, 38, 6, 4].

Motivation The availability of efficient reasoners has proved to be important in both the design and deployment of ontologies. Designing ontologies is an extremely complex task, and modern ontology design tools typically use reasoners to support the ontologist by highlighting inconsistencies in the design and allowing them to compare their intuitions about implicit subsumption relationships between classes with those computed by the reasoner [2, 31]. This kind of reasoning support is, for example, provided by both OilEd and the ProtégéOWL plugin, tools which are increasingly used for ontology development in e-Science.

Applications of ontologies typically involve querying, and this again means using a reasoner, e.g., to determine when an individual (or a tuple of individuals) satisfies a query expression, or to retrieve all individuals (or tuples) satisfying a given query [22, 10]. For example, a biologist may want to answer queries about gene product data annotated with terms from the Gene Ontology, with a reasoner being used to determine which gene products are instances of complex descriptions that also use terms from the ontology [16].

The known decidability of key reasoning problems (such as satisfiability, subsumption and instance retrieval), and the availability of efficient reasoners based on highly optimised tableau decision procedures, were crucial factors in motivating the DL based design of DAML+OIL and OWL [19, 20]. Decidability comes, however, at a cost in terms of restricted expressive power. In particular, while such languages are generally equipped with a relatively rich set of constructors for use with classes (unary predicates), they only provide a very limited set of constructors for use with properties (binary predicates). These limitations can be onerous in some applications, in particular those where aggregation plays a prominent role. For example, in complex physically structured domains such as biology and medicine it is often important to describe structures that are exactly equivalent to the aggregation of their parts, and to have properties of the component parts transfer to the whole (a femur with a fractured shaft is a fractured femur) [32]. The importance of this kind of knowledge can be gauged from the fact that it can invariably be expressed in ontology languages designed specifically for medicine, even those that are otherwise relatively weak [33, 40]; various "work-arounds" have also been described for use with ontology languages that cannot express this kind of knowledge directly [37].

Similarly, in grid and web services applications, it may be necessary to describe composite processes in terms of their component parts, and to express relationships between the properties of the various components and those of the composite process. For example, in a sequential composition of processes it may be useful to express a relationship between the inputs and outputs of the composite and those of the first and last component respectively, as well as relationships between the outputs and inputs of successive components [44].

These limitations can be overcome to some extent by extending the DL language with a so-called *role-box* [21], but in order to maintain decidability it is necessary to impose severe restrictions on what can be expressed. For example, this framework would not allow the expression of simple family relationships such as the fact that "uncle" is equivalent to the composition of "parent" and "brother".

In addition to these problems with domain ontologies, many richly axiomatised foundational ontologies, such as SUMO[1] and DOLCE [29], are based on full FOL with relations of arbitrary arity, and even on extensions of FOL using relations with variable arities. This makes it impossible for DL based tools to exploit these foundational ontologies in order to structure or validate domain ontologies, and to improve interoperability between ontologies.

[1] See http://suo.ieee.org/

A recognition of the limitations of DL based ontology languages, in particular in web services applications, has led to proposals to extend them with, e.g., Horn-clause axioms [17, 18], or even axioms supporting arbitrary use of first order quantification [6, 4, 38]. These extended languages are based on larger fragments (than the DL fragment) of FOL, and may even be equivalent to full FOL; as a consequence, computing class consistency and subsumption is no longer decidable in general.

The utility of such languages, and the applications that use them, will crucially depend on the provision of reasoning support: there is little point in building complex models of web services without any means of manipulating or querying them.

First-Order Theorem Provers First-order theorem provers have traditionally been used for the same purpose as DL-based ontology reasoners: providing reasoning services. They have a long history: indeed, some first-order theorem provers had already been implemented in the 1960s. There are, however, many important differences between FO provers and DL reasoners which explain why FO provers have not yet achieved widespread use in the Semantic Web.

FO provers deal with an undecidable logic. They are highly optimised for general-purpose reasoning, and are especially optimised for reasoning with equality. For example, they can often find very complex combinatorial proofs of identities in algebras. FO provers are based on a highly advanced theory of saturation algorithms with redundancy. This theory is very flexible—for example, completeness theorems in it have been proven for inference systems using arbitrary literal selection functions that can simulate various proof-search strategies, such as bottom-up or top-down reasoning.

Recently, there have been papers showing how FO provers can be used to reason in theories with a rich definitional structure [8, 12]. Experiments into their use for classifying DL-based ontologies, using a naive translation of DL formulas into first-order formulas, have also shown encouraging results [45].

Nonetheless, [45] also shows that on DL-based ontologies using simple languages DL reasoners are much faster than a straightforwardly used FO prover. However, the use of DL reasoners for more expressive languages faces a number of obstacles. For example, different languages need different reasoning algorithms, and an efficient implementation of a new inference algorithm may require the re-implementation of data structures supporting efficient inference procedures. In contrast, FO provers use a well-established uniform inference mechanism with thoroughly investigated implementation techniques and data structures (see, e.g., [39, 35, 36]); tuning them for new applications usually requires only implementation of new preprocessing algorithms, and finding the best settings for a wide range of already available parameters.

Traditionally, FO provers have been used for proving theorems in mathematics, and for software and hardware verification. For these applications the axiomatisation is normally relatively small, and also has a small number of function and predicate symbols. In contrast, ontologies may contain a very large number of axioms and predicate symbols. Moreover, axiomatisations of different theories in FOL offer a great variety of different constructs, while ontologies typically contain many similarly-structured "definitions".

The experimental results in [49] show that the inverse method (a non-tableau method based on a saturation algorithm) can be implemented just as efficiently as tableau-based DL provers. The implementation reported in this paper required less than one second to answer queries to the SUMO ontology, which contains about 5,000 first-order axioms with equality. Moreover, it took only a few seconds to to find a large number of non-trivial inconsistencies in various versions of SUMO and in an even larger terrorism ontology.

The KIF Language KIF is a language for expressing knowledge that contains full first-order logic, and extends it with several features. First, it supports some datatypes, for example real numbers, and evaluable functions on these datatypes. Second, it can use arbitrary terms, including variables, as function and predicate symbols. Third, it has row variables which range over sequences of terms of arbitrary finite lengths. As a consequence, KIF allows for functions and relations of variable arities. The semantics of KIF is described in [14].

Support for datatypes seems to be crucial for many applications, and there are numerous proposals to include datatypes in Semantic Web languages. Full support of datatypes in first-order logic is impossible: having a datatype of integers with simple operations on it means that one can express arithmetic, and therefore there is no hope for the automation of reasoning in first-order logic with datatypes.

The second feature of KIF—variables as function and predicate symbols—is not well-understood. Indeed, it was believed that one could translate this feature in first-order logic by adding a (meta) predicate *holds*, and replacing formulas like $x(t)$ by $holds(x, t)$, but [25] have shown that this is not so.

Concerning row variables, [14] note that they are not first-order, but their full expressive power should be further investigated. The following theorem shows that one can express arithmetic in first-order logic with row variables.

Theorem 1. *There exists a polynomial-time translation τ of arithmetic into predicate logic with row variables such that for every sentence F or arithmetic, F holds in the standard model of arithmetic if and only if $\tau(F)$ is valid in predicate logic with row variables.*

This theorem shows that there is no hope for the automation of reasoning in first-order with arbitrary row variables; in particular, the set of theorems of first-order logic with row variables is not recursively enumerable.

2 Adapting Vampire for Large Ontologies

Query Answering Query answering requires retrieving individuals which satisfy a given formula. That is, given a query $Q(\bar{x})$ with free variables $\bar{x}$, one has to find (all, or a given number of) vectors of terms $\bar{t}$ such that $Q(\bar{t})$ is a logical consequence of formulas in the ontology. To implement query answering, one needs to modify inference algorithms to return individuals satisfying the query. To implement it efficiently against large knowledge bases, one has to, in addition, be able to answer a sequence of queries without reloading the ontology.

Query answering requires retrieving individuals which satisfy a given formula. For example, one can ask the following query to find all individuals who published a paper at FoIKS

```
has_paper(X,'FoIKS')
```

To implement query answering, one needs to modify inference algorithms to return individuals satisfying the formula.

Returning Individuals Database systems perform query answering but not theorem proving. DL reasoners were originally designed for theorem proving, but some of them can now perform restricted forms of query answering as well. For FO provers, query answering is not very different from theorem proving and can be implemented essentially at no extra cost. The standard way of providing query answering is via an answer predicate. For example, for the above query with one variable `X` we introduce a unary answer predicate `answer`, replace the query by the formula

```
has_paper(X,'FoIKS') -> answer(X)
```

and run a standard saturation algorithm with one difference: instead of searching for a derivation of the empty clause (to signal that a refutation is found) we search for derivations of clauses whose only predicate symbol is `answer`. For example, if one derives

```
answer('Andrei'),
```

then `'Andrei'` is an answer to the query. Answer predicates are a very powerful mechanism. They can be used for finding multiple answers, disjunctive answers, or general answers with variables. For example, a derivation of

```
answer('Andrei') \/ answer('Ian'),
```

means that either `'Andrei'` or `'Ian'` satisfy the query (but there may be not enough information for a definite answer). Likewise, having derived `answer(Y)`, where `Y` is a variable, means that every object satisfies the query (which may signal that something is wrong with the ontology or with the query).

Answer predicates are implemented in a number of theorem provers, including Vampire, Otter [28] and Gandalf [43]. For Vampire, one can also specify that only definite answers should be considered. To implement definite answers, Vampire replaces any clause $a(s_1, \ldots, s_n) \vee a(t_1, \ldots, t_n) \vee C$, where a is the answer predicate, by the clause $s_1 \neq t_1 \vee \ldots \vee s_n \neq t_n \vee a(t_1, \ldots, t_n) \vee C$. Completeness of resolution with answer predicates was studied in [42].

Answering Sequences of Queries One essential difference between theorem provers and query answering systems is that the former are normally invoked to solve a single problem. If another problem has to be solved, the theorem prover has to be called again. The cost of activating a query answering system working with a large ontology may be prohibitive. For example, VAMPIRE implements sophisticated preprocessing algorithms for first-order formulas, and the collection of clauses obtained from them can be further processed for simplifications. It is better to have a system which can answer a sequence of queries to an ontology or collection of ontologies without restarts. VAMPIRE solves this problem by implementing *pre-compiled knowledge bases.*

Figure 1 shows a part of VAMPIRE's *bag file*[2] that uses pre-compiled knowledge bases in VAMPIRE. The command `kb_load` reads an ontology or a knowledge base from a file and compiles it. This is done by preprocessing formulas in the ontology, converting them to CNF, and applying various simplification rules to the CNF. The resulting set of clauses is then stored internally, along with some information that allows one to quickly retrieve only a part of the ontology that is relevant to a particular query. The command `kb_status` can be used to enable or disable loaded ontologies (disabled ontologies are not used for query answering but remain stored and pre-compiled). The formulas in the enabled knowledge bases can be used for query answering. Answering some queries may require information from several knowledge bases. As far as we know, VAMPIRE is the only first-order prover implementing pre-compiled knowledge bases.

Goal-Oriented Proof-Search for Query Answering When answering queries to large ontologies, it is of paramount importance that query answering be goal-oriented. One can use the modern theory of resolution (see [1]) to make resolution proof-search goal-oriented. For example, one can use literal selection functions and term orderings that prefer literals and terms coming from the goal. Another possibility is to use the *set-of-support strategy*, in which inferences involving only clauses not derived from the query are prohibited. The use of the set of support strategy together with selection functions is incomplete, so we used the following modification of this strategy: we select *all* literals in clauses not derived from the query. Experimental results have shown that this variant of the set-of-support strategy works very well: a typical query response time for queries to SUMO falls within one second.

We believe that in future one can improve goal-oriented proof search by also providing relevance filters, which will allow one to focus on a small part of the ontology. This seems to be especially promising for ontologies, in which the majority of knowledge is represented as definitions of predicates. Such a filtering technique proved indispensable in the use of theorem provers for classifying ontologies based on the subsumption relation (see [45]).

[2] A bag file may contain any kind of information for VAMPIRE, including commands, options, and queries.

```
<!-- load the terrorism ontology -->
<kb_load kb="terrorism" syntax="kif"
         file="SemWeb/terrorism.kif" />

<!-- answer the following query to the terrorism ontology -->
<query max_answers="10" time_limit="5">
  (instance ?X ?Y)
</query>

<!-- load SUMO and disable the terrorism ontology -->
<kb_load kb="sumo" syntax="kif"
         file="SemWeb/sumo139.kif" />
<kb_status kb="terrorism" enabled="no" />

<!-- answer the following query to SUMO -->
<query max_answers="10" time_limit="5">
  (instance ?X ?Y)
</query>

<!-- enable the terrorism ontology -->
<kb_status kb="terrorism" enabled="yes" />

<!-- answer the same query using formulas from both SUMO
 and the terrorism ontology -->
<query max_answers="10" time_limit="5">
  (instance ?X ?Y)
</query>

<!-- assert two new facts about the instance relation -->
<assert syntax="kif">
  (instance a b)
  (instance b c)
</assert>

<!-- answer the same query using formulas from SUMO,
  the terrorism ontology and newly asserted facts -->
<query max_answers="10" time_limit="5">
  (instance ?X ?Y)
</query>
```

Fig. 1. Pre-compiled Knowledge Bases

Consistency Checking For DL-based ontologies, consistency (of the ontology as a whole) is usually not an issue. For ontologies and knowledge bases using expressive languages, such as FOL, consistency may be a problem. Such ontologies may be created by people using the same symbols with a different meaning, people who do not know logic well, or people who understand the ontology domain differently. Our experiments with several versions of SUMO [30] have shown that all of them had numerous axioms creating inconsistency. Two examples (one of them for a terrorism ontology) are given in the appendix.

Checking consistency of ontologies is a more difficult problem than query answering. For query answering one can focus on the query and formulas derived from it. For consistency-checking, there is no query to focus on. Consistency checking in Vampire was implemented by a standard saturation algorithm. However it turned out that the standard options used for theorem proving did not perform well for large ontologies. (This may be one of the reasons for the relatively slow performance of Vampire reported in [45].) By turning off some simplification rules (backward subsumption, backward demodulation, forward subsumption resolution) and fine-tuning some other options, we were able to increase the speed by a factor of 12.

Beyond First-Order Logic Even relatively simple extensions of both DLs and FOL can be very difficult to implement or even lead to theories for which no complete algorithms exist. For example, it is not hard to achieve the full expressive power of arithmetic by adding integers as a datatype and using this datatype in an unrestricted way. Nonetheless, extensions of first-order logic may turn out to be crucial for applications, and in this case one has to find a compromise between the expressive power of such extensions and the possibility of implementing them efficiently on top of existing implementations. In this section we describe a light-weight implementation in Vampire of some features taking the system beyond first-order logic. We illustrate the utility of such extensions by showing inconsistency proofs for ontologies using these features.

Support for Datatypes Theorem provers are not able to deal with datatypes (such as integers and strings) other than via complex axiomatisations that would severely degrade performance. Typical ontologies, particularly in web services applications, will contain many datatypes and data values recording knowledge about names, addresses, prices and so on. Full support of these datatypes in a prover is impossible since first-order logic with datatypes is not recursively enumerable. However, a limited form of reasoning with datatypes can be implemented.

Vampire supports three datatypes: integers, real numbers and strings. It can understand constants of these datatypes occurring in the input ontology, for example it knows that 1 is an integer constant and "one" is a string constant. It also implements a number of built-in functions and relations on these datatypes, such as comparison operators on numeric datatypes or concatenation of strings. Vampire can evaluate simple expressions. For example, it can evaluate $2+3 < 6$

to `true`. It cannot do more complex reasoning tasks with the datatypes, for example, it cannot to derive a contradiction from the facts $c < 2$ and $3 < c$, but will be able derive a contradiction from them if the input contains the transitivity axiom for $<$.

Moreover, VAMPIRE has a mechanism that allows one to define new functions and relations on the datatypes using recursive definitions, so in a way it contains a small functional programming language inside. Such definitions can be implemented using pre-oriented equalities and pre-selected literals. The ability to define new functions and relations on datatypes can become standard in future expressive ontology languages.

Since there is no standard convention on the names for built-in functions and relations on datatypes, VAMPIRE also provides an interface for mapping the input ontology names to the internal names for these functions and relations.

Meta-Predicates It is non-trivial to implement variables and functions as predicate symbols, as there is no proof theory for this feature of KIF. If VAMPIRE finds such variables in the input, it transforms the input using the predicate symbol *holds*, as mentioned in Section 1, and the function symbol *apply*. This style of reasoning is incomplete, since VAMPIRE does not implement *reflection*, that is, the rule replacing $holds(r, t_1, \ldots, t_n)$ by $r(t_1, \ldots, t_n)$ or vice versa. Even the current lightweight implementation using *holds* was strong enough to answer some queries related to transitive relations axiomatised by

$$transitive(u) \leftrightarrow \forall x \forall y \forall z (u(x, y) \wedge u(y, z) \rightarrow u(x, z))$$

and to discover inconsistencies in SUMO, see the appendix.

Row Variables Theorem 1 implies the impossibility of reasoning with row variables. If all row variables are bound by essentially universal quantifiers (that is, positively occurring universal or negatively occurring existential ones), then the set of provable formulas is still recursively enumerable but one has to implement reasoning modulo associativity, see [14]. Implementing reasoning modulo associativity is very difficult, for example, two terms may have an infinite number of minimal unifiers modulo associativity. It is pointed out in [14] that one can use a weaker class of formulas with row variables in which a row variable may occur only as the last argument. In this case it is enough to implement only *sequence variables* [26].

However, one can note that the main use of row variables in SUMO is for relations or relatively small arities. For this reason, VAMPIRE only substitutes for row variable sequences of variables of a bounded length. The default upper bound on the length is 2 but it can be changed by the user.

If the input contains row variables, VAMPIRE does the following:

1. Reject formulas that contain row variables bound by essentially existential quantifiers;
2. Substitute every remaining row variable @ by sequences or ordinary variables $x_1, \ldots, x_i$, such that i ranges over $0, \ldots, n$ where n is the upper bound specified by the user. This rule is called *row variable expansion.*

Appendix A gives an example involving reasoning with row variables. The latest version of SUMO contains only two formulas rejected by VAMPIRE.

Other issues

Proof Output It is important that an answer to a query comes with an explanation. Likewise, when inconsistency is discovered, one needs an explanation to find a source of inconsistency. One can also require that the answers provided by an ontology reasoner could be checked by a proof checker. To this end, one needs a system able to produce proofs. Most of the currently available theorem provers produce a proof in some form. VAMPIRE can produce proofs in several formats, including XML. Our experiments on checking consistency of SUMO have shown that the proof should be understandable by humans. Indeed, when an inconsistency is discovered, one should find the axioms that cause inconsistency and repair the ontology to remove all sources of inconsistency. We have found that the proof format in which each inference is displayed separately is easier to read and understand. Moreover, an ASCII proof is not very readable, so we included an option to output proofs in LaTeX. Both proofs in the appendix have been generated by VAMPIRE automatically and slightly edited to fit in the paper size.

The output proof format of VAMPIRE is still far from perfect. We believe that further research should be done to improve presentation of complex computer-generated proofs in a human-friendly form.

We are currently working on producing proofs checkable in $O(n \log n)$ time. These proofs will be more detailed than the human-readable proofs but the ability to check proofs by using a proof-checker is indispensable both in debugging the theorem prover, the use of the prover for safety-critical applications and automatic analysis of proofs.

3 Future Work

The techniques we have described greatly improve the performance of VAMPIRE when answering queries to and checking the consistency of ontologies. Like other resolution theorem provers, however, it is not very effective at proving satisfiability, and hence at proving non-subsumption. This is a problem if VAMPIRE is to be used for general purpose ontology reasoning: in typical ontologies, for example, most classes are satisfiable and most pairs of concepts are not in a subsumption relationship.

Future work will, therefore, include investigations of a number of strategies for addressing this problem. Firstly, we will investigate improved literal selection strategies (that exploit the structure of ontology axioms) to improve the performance of VAMPIRE on satisfiable problems. Secondly, we will investigate the enhancement of existing model building methods (see, e.g., [24, 27, 7]), which are designed to prove satisfiability in FOL. The idea here is to use similar techniques to those we have already successfully employed in VAMPIRE, in particular using relevance filters to reduce the effective size of the ontology and exploiting the special structure of ontology axioms (in this case to try to minimise the problem of exponential explosion in model size). Finally, for suitable undecidable extensions of DLs (e.g., the proposed SWRL Horn-clause extension to OWL [17, 18]), we will investigate the development of model building algorithms based on existing tableaux decision procedures for DLs. Such an algorithm would still be sound for satisfiability (i.e., it would only succeed in building a model if the problem is satisfiable), but it will no longer be guaranteed to terminate.

We believe that a combination of some or all of these satisfiability testing techniques with a suitably optimised resolution prover (such as VAMPIRE) will be able to solve the vast majority of problems encountered when reasoning with ontologies.

References

1. L. Bachmair and H. Ganzinger. Resolution theorem proving. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 2, pages 19–99. Elsevier Science, 2001.
2. S. Bechhofer, I. Horrocks, C. Goble, and R. Stevens. OilEd: A reason-able ontology editor for the semantic web. In F. Baader, G. Brewka, and T. Eiter, editors, *KI 2001: Advances in Artificial Intelligence*, volume 2174 of *Lecture Notes in Computer Science*, pages 396–408. Springer Verlag, 2001.
3. S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. OWL web ontology language 1.0 reference. W3C Recommendation, 10 February 2004. Available at http://www.w3.org/TR/owl-ref/.
4. D. Berardi, M. Grüninger, R. Hull, and S. McIlraith. Towards a first-order ontology for semantic web services. http://www.w3.org/2004/08/ws-cc/mci-20040904, sep 2004.
5. T. Berners-Lee, J. Hendler, and O. Lassila. The semantic Web. *Scientific American*, 284(5):34–43, 2001.
6. H. Boley, M. Dean, B. Grosof, M. Sintek, B. Spencer, S. Tabet, and G. Wagner. First-Order-Logic RuleML. http://www.ruleml.org/fol/, 2004.
7. K. Claessen and N. Sörensson. New techniques that improve mace-style model finding. In *Proceedings of the Workshop Model Computation 2003*, 2003.
8. A. Degtyarev, R. Nieuwenhuis, and A. Voronkov. Stratified resolution. *Journal of Symbolic Computations*, 36(1-2):79–99, 2003.
9. A. Emmen. The grid needs ontologies—onto-what?, 2002. http://www.hoise.com/primeur/03/articles/monthly/AE-PR-02-03-7.html.
10. R. Fikes, P. Hayes, and I. Horrocks. OWL-QL—a language for deductive query answering on the Semantic Web. *Journal of Web Semantics*, 2004. To Appear.
11. I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration, 2002. http://www.globus.org/research/papers/ogsa.pdf.
12. H. Ganzinger and J. Stuber. Superposition with equivalence reasoning and delayed clause normal form transformation. In F. Baader, editor, *19th International Conference on Automated Deduction (CADE-19)*, volume 2741 of *Lecture Notes in Computer Science*, pages 335–349. Springer Verlag, 2003.

13. V. Haarslev and R. Möller. *RACER User's Guide and Reference Manual. Version 1.7.7*, Sept. 2003.
14. P. Hayes and C. Menzel. A semantics for the knowledge interchange format. In *IJCAI 2001 Workshop on the IEEE Standard Upper Ontology*, 2001.
15. I. Horrocks. Using an expressive description logic: FaCT or fiction? In A. Cohn, L. Schubert, and S. Shapiro, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR'98)*, pages 636–647, San Francisco, CA, June 1998. Morgan Kaufmann.
16. I. Horrocks, L. Li, D. Turi, and S. Bechhofer. The instance store: DL reasoning with large numbers of individuals. In *Proc. of the 2004 Description Logic Workshop (DL 2004)*, pages 31–40, 2004.
17. I. Horrocks and P. F. Patel-Schneider. A proposal for an OWL rules language. In *Proc. of the Thirteenth International World Wide Web Conference (WWW 2004)*, pages 723–731. ACM, 2004.
18. I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A semantic web rule language combining owl and ruleml. W3C Note, 21 May 2004. Available at http://www.w3.org/Submission/SWRL/.
19. I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. Reviewing the design of DAML+OIL: An ontology language for the semantic web. In *Proc. of the 18th National Conference on Artificial Intelligence (AAAI 2002)*, pages 792–797. AAAI Press, 2002.
20. I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003.
21. I. Horrocks and U. Sattler. The effect of adding complex role inclusion axioms in description logics. In *Proc. of the 18th Int. Joint Conf. on Artificial Intelligence (IJCAI 2003)*, pages 343–348. Morgan Kaufmann, 2003.
22. I. Horrocks and S. Tessaris. Querying the semantic web: a formal approach. In I. Horrocks and J. Hendler, editors, *First International Semantic Web Conference (ISWC 2002)*, number 2342 in Lecture Notes in Computer Science, pages 177–191. Springer Verlag, 2002.
23. U. Hustadt, B. Konev, A. Riazanov, and A. Voronkov. TeMP: A temporal monodic prover. In M. R. D.A. Basin, editor, *Automated Reasoning - Second International Joint Conference, IJCAR 2004*, volume 3097 of *Lecture Notes in Computer Science*, pages 326–330. Springer Verlag, 2004.
24. U. Hustadt and R. A. Schmidt. Using resolution for testing modal satisfiability and building models. In I. P. Gent, H. van Maaren, and T. Walsh, editors, *SAT 2000: Highlights of Satisfiability Research in the Year 2000*, volume 63 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, Amsterdam, 2000. Also to appear in a special issue of *Journal of Automated Reasoning*.
25. P. P.-S. I. Horrocks. Three theses of representation in the semantic web. In *Proceedings of the Twelfth International World Wide Web Conference, WWW2003*, pages 39–47, Budapest, Hungary, Jan. 2003. ACM.
26. T. Kutsia. Theorem proving with sequence variables and flexible arity symbols. In M.Baaz and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR 2002)*, volume 2514 of *Lecture Notes in Artificial Intelligence*, pages 278–291, Tbilisi, Georgia, 2002.
27. W. McCune. Mace4 reference manual and guide. Technical Memorandum 264, Argonne National Laboratory, Aug. 2003.
28. W. McCune. OTTER 3.3 reference manual. Technical Memorandum 263, Argonne National Laboratory, Aug. 2003.

29. P. Mika, D. Oberle, A. Gangemi, and M. Sabou. Foundations for service ontologies: Aligning OWL-S dolce. In S. Feldman, M. Uretsky, M. Najork, and C. Wills, editors, *WWW 2004, Proceedings of the 13th international conference on World Wide Web*, pages 563–572. ACM, 2004.
30. A. Pease, I. Niles, and J. Li. The Suggested Upper Merged Ontology: A large ontology for the Semantic Web and its applications. In *Working Notes of the AAAI-2002 Workshop on Ontologies and the Semantic Web*, Edmonton, Canada, 2002.
31. Protégé. http://protege.stanford.edu/, 2003.
32. A. Rector. Analysis of propagation along transitive roles: Formalisation of the galen experience with medical ontologies. In *Proc. of DL 2002*. CEUR (http://ceur-ws.org/), 2002.
33. A. Rector, S. Bechhofer, C. A. Goble, I. Horrocks, W. A. Nowlan, and W. D. Solomon. The GRAIL concept modelling language for medical terminology. *Artificial Intelligence in Medicine*, 9:139–171, 1997.
34. A. Rector and I. Horrocks. Experience building a large, re-usable medical ontology using a description logic with transitivity and concept inclusions. In *Proc. of the 13th Nat. Conf. on Artificial Intelligence (AAAI 97)*, 1997.
35. A. Riazanov and A. Voronkov. The design and implementation of Vampire. *AI Communications*, 15(2-3):91–110, 2002.
36. A. Riazanov and A. Voronkov. Limited resource strategy in resolution theorem proving. *Journal of Symbolic Computations*, 36(1-2):101–115, 2003.
37. S. Schulz and U. Hahn. Parts, locations, and holes - formal reasoning about anatomical structures. In *Proc. of AIME 2001*, volume 2101 of *Lecture Notes in Artificial Intelligence*. Springer Verlag, 2001.
38. Common Logic Standard. http://cl.tamu.edu/.
39. R. Sekar, I. Ramakrishnan, and A. Voronkov. Term indexing. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume II, chapter 26, pages 1853–1964. Elsevier Science, 2001.
40. K. Spackman. Managing clinical terminology hierarchies using algorithmic calculation of subsumption: Experience with snomed-rt. *J. of the Amer. Med. Informatics Ass.*, 2000. Fall Symposium Special Issue.
41. R. Stevens, C. Goble, I. Horrocks, and S. Bechhofer. Building a bioinformatics ontology using OIL. *IEEE Transactions on Information Technology in Biomedicine*, 6(2):135–141, 2002.
42. T. Tammet. Completeness of resolution for definite answers. *Journal of Logic and Computation*, 5(4):449–471, 1995.
43. T. Tammet. Gandalf. *Journal of Automated Reasoning*, 18(2):199–204, 1997.
44. The DAML Services Coalition. DAML-S: Web service description for the semantic web. In *Proc. of the 2003 International Semantic Web Conference (ISWC 2003)*, number 2870 in Lecture Notes in Computer Science. Springer Verlag, 2003.
45. D. Tsarkov, A. Riazanov, S. Bechhofer, and I. Horrocks. Using Vampire to reason with OWL. In *Semantic Web 2004*. Springer Verlag, 2004. to appear.
46. S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, and P. Vanderbilt. Grid service specification (draft). GWD-I draft , GGF Open Grid Services Infrastructure Working Group, 2002.
47. M. Uschold, M. King, S. Moralee, and Y. Zorgios. The enterprise ontology. *Knowledge Engineering Review*, 13, 1998.
48. F. van Harmelen, P. F. Patel-Schneider, and I. Horrocks. Reference description of the DAML+OIL (March 2001) ontology markup langauge, Mar. 2001.

49. A. Voronkov. $k\mathcal{E}$: a theorem prover for K. In H. Ganzinger, editor, *Automated Deduction—CADE-16. 16th International Conference on Automated Deduction*, volume 1632 of *Lecture Notes in Artificial Intelligence*, pages 383–387, Trento, Italy, July 1999.

A Appendix

In this appendix we give two inconsistency proofs found by VAMPIRE. Each of the proofs is explained in a separate section. We have not changed the formulation of the problems but do not present them in the KIF syntax. The proofs were produced using the LaTeX output facility of VAMPIRE. We had to edit them slightly to fit in the paper size. We also renamed several functions and relations in SUMO for a better readability using the renaming feature provided by VAMPIRE. For example, we write $x : y$ instead of $\mathit{instance}(x, y)$. A proof consists of a sequence of inferences. Each inference infers a formula, called the *conclusion* of an inference from zero or more formulas, called the *premises* of the inference. All formulas occurring in the proof are numbered. Each inference is annotated by the numbers of the premises, the number of conclusion, and inference rules used in the inference. For example, the annotation of last inference in the first proof means that formula 9 was obtained from formulas 1,2 and 8 using the resolution and forward subsumption resolution inference rules. The symbol $\Box$ denotes the empty clause, which is logically equivalent to contradiction.

We have many examples of inconsistency proofs found by VAMPIRE in the latest versions of SUMO and the ontology of terrorism from the Sumo Web page http://ontology.teknowledge.com. A typical proof occupies at least two pages, so we cannot give them here in detail.

We give a very brief account of one (very short) proof of inconsistency of the ontology of terrorism. This ontology has about 18,000 first-order axioms. This example illustrates why most (if not all) of the previous provers would not be able to find this inconsistency. Although the proof is relatively short, VAMPIRE generates over 60,000 formulas to find it.

Essentially, the proof is based on three axioms in the ontology. Two of the axioms are facts asserting information about the number of victims of two Hamas attacks. The problem with the axiomatisation of these attacks comes from the fact that they were given the same name.

$$\begin{array}{l}\mathit{victimDeathCount}(\mathit{HAMAS\text{-}KnifeAttack\text{-}25\text{-}Jun\text{-}92}, 0);\\ \mathit{victimDeathCount}(\mathit{HAMAS\text{-}KnifeAttack\text{-}25\text{-}Jun\text{-}92}, 2).\end{array}$$

These two axioms look contradictory, but they do not contradict each other in FOL. However, the ontology also contains the following axiom

$$\begin{array}{l}\mathit{victimDeathCount}(x_0, x_3) \supset\\ x_3 = \mathit{CardinalityFn}(\mathit{KappaFn}(x_4, \mathit{and}(\mathit{patient}(x_0, x_4),\\ \qquad\qquad \mathit{holdsDuring}(\mathit{ImmediateFutureFn}(x_0),\\ \qquad\qquad\qquad\qquad \mathit{attribute}(x_4, \mathit{Dead}))))).\end{array}$$

This axiom looks quite complex, nonetheless it is easy to see that it implies that in the relation *victimDeathCount* the second argument is a function of the first argument. This and the two facts given above imply that $0 = 2$. Since VAMPIRE knows simple arithmetic, it immediately derives contradiction. The proof is found in 1.7 seconds on a computer with a 1GHz Intel processor and 2GB of RAM.

The proof is neither long nor very sophisticated. However, to find it one has to apply the equality rule paramodulation to rather complex terms used in the last formula. This would make it very difficult if possible at all to find it for a prover not having efficient built-in equality reasoning or a prover using a translation of logic with equality into logic without equality. In addition, knowledge of simple arithmetic is needed to derive contradiction from $0 = 2$, and as far as we know the most efficient FO provers do not have built-in arithmetic in any form.

We have a large collection of proofs of inconsistency of several versions of SUMO found by VAMPIRE; some of them, if translated into human proofs, use very refined argument, for example showing problems with a careless use in SUMO of row variables and relations of arbitrary arity. We are planning to analyse these proofs in a separate paper.

A Proof of Inconsistency of SUMO Here we give a proof demonstrating inconsistency of the Suggested Upper Merger Ontology version of July 2004. This proof is generated by the auto-mode of VAMPIRE in 34.5 seconds. VAMPIRE can also find the inconsistency proof in 3 seconds with a time limit of 4 seconds.[3] Using the optimal settings for ontology reasoning VAMPIRE can prove inconsistency in 2.7 seconds, and also in 0.7 second with a time limit of 1 second. If incomplete strategies are used, the proof can be found in 0.2 seconds. Note that consistency checking is much harder than query answering, since there is no goal to focus on, so a theorem prover must perform a brute force non-goal-oriented search from the initial set of about 5,000 FO formulas with equality. This proof derives contradiction from a formula containing a row variable and uses the row variable expansion rule. A row variables occurs, for example, in the atomic subformula $holds(x, @_1, w)$ of input formula 5.

The proof is written in a rather condensed form. For example, the CNF transformation rule in the proof consists of a number of smaller steps, including skolemisation introducing the skolem function σ.

Proof.

[1, input]

$$ListFn : TotalValuedRelation$$

[2, input]

$$ListFn : VariableArityRelation$$

[3] VAMPIRE may work much faster when a time limit is given, for details see [36].

[3, input]

$$x : \mathit{VariableArityRelation} \supset \neg(\exists y)\mathit{valence}(x, y)$$

[3 → 4, cnf transformation]

$$\begin{array}{c} x : \mathit{VariableArityRelation} \supset \neg(\exists y)\mathit{valence}(x, y) \\ \hline \neg \mathit{valence}(x, y) \vee \neg x : \mathit{VariableArityRelation} \end{array}$$

[5, input]

$$\begin{array}{l} x : \mathit{TotalValuedRelation} \equiv \\ \quad (\exists y)(\, x : \mathit{Relation} \wedge \mathit{valence}(x, y) \wedge \\ \qquad\quad (\,(\forall z \forall u \forall v)(z < y \wedge \mathit{domain}(x, z, v) \wedge u = \mathit{nth}(\mathit{ListFn}(@_1), z) \supset u : v) \supset \\ \qquad\qquad (\exists w)\mathit{holds}(x, @_1, w))) \end{array}$$

[5 → 6, row variable expansion]

$$\begin{array}{l} x : \mathit{TotalValuedRelation} \equiv \\ \quad (\exists y)(\, x : \mathit{Relation} \wedge \mathit{valence}(x, y) \wedge \\ \qquad\quad (\,(\forall z \forall u \forall v)(z < y \wedge \mathit{domain}(x, z, v) \wedge u = \mathit{nth}(\mathit{ListFn}(@_1), z) \supset u : v) \supset \\ \qquad\qquad (\exists w)\mathit{holds}(x, @_1, w))) \\ \hline x : \mathit{TotalValuedRelation} \equiv \\ \quad (\exists y)(\, x : \mathit{Relation} \wedge \mathit{valence}(x, y) \wedge \\ \qquad\quad (\,(\forall z \forall u \forall v)(z < y \wedge \mathit{domain}(x, z, v) \wedge u = \mathit{nth}(\mathit{ListFn}(w), z) \supset u : v) \supset \\ \qquad\qquad (\exists x_6)\mathit{holds}(x, w, x_6))) \end{array}$$

[6 → 7, cnf transformation]

$$\begin{array}{l} x : \mathit{TotalValuedRelation} \equiv \\ \quad (\exists y)(\, x : \mathit{Relation} \wedge \mathit{valence}(x, y) \wedge \\ \qquad\quad (\,(\forall z \forall u \forall v)(z < y \wedge \mathit{domain}(x, z, v) \wedge u = \mathit{nth}(\mathit{ListFn}(w), z) \supset u : v) \supset \\ \qquad\qquad (\exists x_6)\mathit{holds}(x, w, x_6))) \\ \hline \qquad\qquad\qquad \mathit{valence}(y, \sigma(x, y)) \vee \neg y : \mathit{TotalValuedRelation} \end{array}$$

[4, 7 → 8, resolution]

$$\begin{array}{c} \neg \mathit{valence}(x, y) \vee \neg x : \mathit{VariableArityRelation} \\ \mathit{valence}(y, \sigma(x, y)) \vee \neg y : \mathit{TotalValuedRelation} \\ \hline \neg y : \mathit{VariableArityRelation} \vee \neg y : \mathit{TotalValuedRelation} \end{array}$$

[1, 2, 8 → 9, resolution, forward subsumption resolution]

$$\begin{array}{c} \mathit{ListFn} : \mathit{TotalValuedRelation} \\ \mathit{ListFn} : \mathit{VariableArityRelation} \\ \neg y : \mathit{VariableArityRelation} \vee \neg y : \mathit{TotalValuedRelation} \\ \hline \Box \end{array}$$

A Proof of Inconsistency of the Terrorism Ontology Here we give a proof of inconsistency of the ontology of terrorism from the Sumo Web page `http://ontology.teknowledge.com`. This ontology has about 18,000 first-order axioms. The proof is rather short but requires knowledge of the datatype of integers. Namely, at the last inference step it uses the fact $0 \neq 2$. In addition, the proof uses built-in equality reasoning. The proof is found in 7 seconds using the standard mode and in 1.7 seconds using the optimal settings for ontology reasoning. Note that the proof uses applications of equality to large terms, so it is unlikely to be found quickly by a prover without built-in equality reasoning.

Proof.

[1, input]

$$\begin{array}{l} \mathit{victimDeathCount}(x,y) \supset \\ \quad y = \mathit{CardinalityFn}(\mathit{KappaFn}(z, \mathit{and}(\mathit{patient}(x,z), \\ \qquad\qquad \mathit{holdsDuring}(\mathit{ImmediateFutureFn}(x), \\ \qquad\qquad\qquad \mathit{attribute}(z, \mathit{Dead}))))) \end{array}$$

[1 → 2, cnf transformation]

$$\frac{\begin{array}{l} \mathit{victimDeathCount}(x,y) \supset \\ \quad y = \mathit{CardinalityFn}(\mathit{KappaFn}(z, \mathit{and}(\mathit{patient}(x,z), \\ \qquad\qquad \mathit{holdsDuring}(\mathit{ImmediateFutureFn}(x), \\ \qquad\qquad\qquad \mathit{attribute}(z, \mathit{Dead}))))) \end{array}}{\begin{array}{l} \quad y = \mathit{CardinalityFn}(\mathit{KappaFn}(x, \mathit{and}(\mathit{patient}(z,x), \\ \qquad\qquad \mathit{holdsDuring}(\mathit{ImmediateFutureFn}(z), \\ \qquad\qquad\qquad \mathit{attribute}(x, \mathit{Dead}))))) \vee \\ \neg \mathit{victimDeathCount}(z,y) \end{array}}$$

[3, input]

$$\mathit{victimDeathCount}(\mathit{HAMAS\text{-}KnifeAttack\text{-}25\text{-}Jun\text{-}92}, 0)$$

[2, 3 → 4, resolution]

$$\frac{\begin{array}{l} \quad y = \mathit{CardinalityFn}(\mathit{KappaFn}(x, \mathit{and}(\mathit{patient}(z,x), \\ \qquad\qquad \mathit{holdsDuring}(\mathit{ImmediateFutureFn}(z), \\ \qquad\qquad\qquad \mathit{attribute}(x, \mathit{Dead}))))) \vee \\ \neg \mathit{victimDeathCount}(z,y) \\ \qquad \mathit{victimDeathCount}(\mathit{HAMAS\text{-}KnifeAttack\text{-}25\text{-}Jun\text{-}92}, 0) \end{array}}{\begin{array}{l} \mathit{CardinalityFn}(\mathit{KappaFn}(y, \mathit{and}(\mathit{patient}(\mathit{HAMAS\text{-}KnifeAttack\text{-}25\text{-}Jun\text{-}92}, y), \\ \quad \mathit{holdsDuring}(\mathit{ImmediateFutureFn}(\mathit{HAMAS\text{-}KnifeAttack\text{-}25\text{-}Jun\text{-}92}), \\ \qquad \mathit{attribute}(y, \mathit{Dead}))))) = 0 \end{array}}$$

[5, input]

$$\mathit{victimDeathCount}(\mathit{HAMAS\text{-}KnifeAttack\text{-}25\text{-}Jun\text{-}92}, 2)$$

[4, 2, 5 → 6, resolution, forward demodulation, evaluation]

$$\frac{\begin{array}{l} \mathit{CardinalityFn}(\mathit{KappaFn}(y, \mathit{and}(\mathit{patient}(\mathit{HAMAS\text{-}KnifeAttack\text{-}25\text{-}Jun\text{-}92}, y), \\ \quad \mathit{holdsDuring}(\mathit{ImmediateFutureFn}(\mathit{HAMAS\text{-}KnifeAttack\text{-}25\text{-}Jun\text{-}92}), \\ \qquad \mathit{attribute}(y, \mathit{Dead}))))) = 0 \\ \quad y = \mathit{CardinalityFn}(\mathit{KappaFn}(x, \mathit{and}(\mathit{patient}(z,x), \\ \qquad\qquad \mathit{holdsDuring}(\mathit{ImmediateFutureFn}(z), \\ \qquad\qquad\qquad \mathit{attribute}(x, \mathit{Dead}))))) \vee \\ \neg \mathit{victimDeathCount}(z,y) \\ \qquad \mathit{victimDeathCount}(\mathit{HAMAS\text{-}KnifeAttack\text{-}25\text{-}Jun\text{-}92}, 2) \end{array}}{\square}$$