WSML Deliverable

D16.2 v0.2
WSML Reasoner Survey

WSML Working Draft – November 16, 2005

Authors:
Jos de Bruijn
Cristina Feier
Uwe Keller
Rubén Lara
Axel Polleres
Livia Predoiu[a]

[a]In alphabetic order

Editors:
Livia Predoiu
Axel Polleres

Reviewers:
Holger Lausen

This version:
`http://www.wsmo.org/TR/d16/d16.2/v0.2/20050902/`
Latest version:
`http://www.wsmo.org/TR/d16/d16.2/v0.2/`
Previous version:
`http://www.wsmo.org/TR/d16/d16.2/v0.2/20050714/`

Abstract

This deliverable provides a survey of different reasoner implementations and discusses their applicability as reasoners with ontologies written in either of the five WSML variants. At first, the requirements each of the WSML variants have on a reasoner are described. Afterwards, we survey interesting and promising reasoner implementations with regard to their applicability on ontologies written in either of the five WSML-variants. The reasoners surveyed in this deliverable range from logic programming engines to description logic reasoners and first-order theorem provers.

Contents

1 Introduction

WSML [de Bruijn et al., 2005] is a family of representation languages for the Semantic Web, taking Description Logics [Baader et al., 2003], Logic Programming [Lloyd, 1987] and First-Order Logic [Fitting, 1996] as semantic basis, with influences from F-Logic [Kifer et al., 1995] and frame-based representation systems.

This deliverable aims at providing a survey on existing reasoner implementations in the context of their applicability in providing core reasoning facilities for the variants of WSML. An existing reasoner implementation can be used for WSML through a wrapper that wraps WSML expressions into the appropriate syntax of the WSML reasoner.

Here, we first survey different reasoner implementations of different logical formalisms ranging from Description Logics and Logic Programming reasoners to full First-Order reasoner implementations and evaluate the usability of the implementations for the different WSML variants. Then, we refer to WSML reasoner implementations that have been developed in the course of the WSML working group and on related developments.

This document is further structured as follows. In Chapter 2 we briefly discuss the different WSML variants and their requirements on a reasoning engine. We present the survey on reasoning implementations in Chapter 3. Finally, we present conclusions and summarize future work and related developments in Chapter 4.

2 WSML Variants

In this chapter we provide a short description of each of the WSML variants, as well as requirements each variant has on the reasoner.

2.1 WSML-Core

WSML-Core is based on the well-known DHL (Description Horn Logic) fragment [Grosof et al., 2003; de Bruijn et al., 2004a], which is that subset of the Description Logic logic $\mathcal{SHIQ}(\mathbf{D})$ (which is close to the language underlying OWL [Horrocks and Patel-Schneider, 2003]) which falls inside the Horn logic fragment of First-Order Logic without equality and without existential quantification.

Two different types of reasoning can be done with WSML-Core, namely: (1) subsumption reasoning and (2) query answering. Subsumption reasoning is equivalent to checking entailment of non-ground formulae and can thus be reduced to checking satisfiability using a First-Order style or a Description Logic-style calculus. Query answering is equivalent to checking entailment of ground facts. Note that query answering in this context boils down to instance retrieval. Query answering can be reduced to satisfiability checking. However, using a First-Order or Description Logic calculus for query answering or instance retrieval is not very efficient [de Bruijn et al., 2004a; Hustadt et al., 2004a]. Fortunately, there are well-known techniques for query answering in the area of logic programming [Lloyd, 1987] and deductive databases [Ullman, 1988].

For subsumption reasoning, the following are the requirements on a reasoner for WSML-Core:

- Subsumption reasoning for WSML-Core can be done through unsatisfiability with Tableaux reasoning, theorem proving or any other technique for checking satisfiability of First-Order theories or query containment[1].
- In order to handle datatypes in WSML-Core, a datatype oracle is required which has a sound and complete procedure for deciding satisfiability of conjunctions of datatype predicates. Such requirements are described in [Pan and Horrocks, 2004].

For query answering, the following are the requirements on a reasoner for WSML-Core:

- A Datalog engine which can handle integrity constraints (for checking datatypes).
- Built-in predicates for handling strings and integers. For integers, basic arithmetic functions (+, -, /, *) should be provided, as well as basic

[1] While query containment for logic programs is in general undecidable, some for restricted forms of logic programs containment can be decided. The simplest form, checking containment of conjunctive queries is well-known to be NP-complete [Chandra and Merlin, 1977]. In [Calvanese et al., 1998; Calvanese et al., 2003] several more expressive query classes and methods to decide query containment for these are discussed. We are however currently not aware of any out-of-the-box implementations for such complex checks.

comparison operators ($=$, $\neq$, $>$, $<$). For strings, at least the (in)equality predicates should be built-in.

- The symbols *true* and *false*. The former represents universal truth; the latter represents falsehood. If *false* is derived form the program, the program is inconsistent. These symbols can be eliminated through simple preprocessing steps (cf. [de Bruijn et al., 2004b]).

2.2 WSML-Flight

WSML-Flight extends WSML-Core with the full expressive power of safe Datalog rules, default negation, the use of integrity constraints (note that constraints are already in WSML-Core; however, they are limited to datatype predicates), (in)equality for abstract individuals, and meta-modeling.

The semantics of WSML-Flight is grounded in Logic Programming. Since there exist no efficient implementation of query containment and since this problem is undecidable in general, the only reasoning task we envision for WSML-Flight is query answering (i.e., entailment of ground facts). Notice that subsumption reasoning can always be done for the WSML-Core subset of an ontology.

The additional requirements for a WSML-Flight reasoner over the requirements for a WSML-Core reasoner are:

- Full Datalog support
- Support for (stratified) default negation
- A built-in equality predicate (in the body of the rule)

Equality could also be axiomatized in the program. However, this would seriously degrade performance of query answering. Therefore we state this as a formal requirement.

The other features added in WSML-Flight compared with WSML-Core can be eliminated in a preprocessing step. However, it would be favorable to have also the following features in the reasoner:

- *Meta-modeling* (treating classes as instances, etc.) can be translated to plain Datalog (cf. [de Bruijn et al., 2004a; Hustadt et al., 2004a; Fensel et al., 2000]). However, if meta-modeling were built into the reasoner, less effort is required in the preprocessing step. Also, query answers might have to be rewritten in order to deal with meta modeling (e.g. [Fensel et al., 2000]).

2.3 WSML-Rule

WSML-Rule extends WSML-Flight with function symbols and additionally allows unsafe rules. Furthermore, WSML-Rule allows unstratified negation under the Well-Founded Semantics [Gelder et al., 1988].

2.4 WSML-DL

WSML-DL is based on the Description Logic $\mathcal{SHIQ}(\mathbf{D})$. Thus, both query answering and subsumption can be done with any of the currently popular DL reasoners, such as FaCT++ [2], RACER [3], and Pellet [4].

Both WSML-Core and WSML-DL are based on the $\mathcal{SHIQ}(\mathbf{D})$ Description Logic. However, WSML-Core corresponds with a restricted subset of $\mathcal{SHIQ}(\mathbf{D})$ which falls in the Horn fragment. Thus, WSML-DL adds the following features to WSML-Core: disjunction, (classical) negation and existential quantification. In terms of complexity, we know that the upper bound for the combined complexity for WSML-Core (not taking into account the datatypes) is in ExpTime [Dantsin et al., 2001], because WSML-Core is a subset of Datalog. Because WSML-Core is strictly less expressive than Datalog, it might be possible to find a lower upper bound. However, we are not aware of any research in this area.

2.5 WSML-Full

The semantics of WSML-Full has not yet been defined. However, it is envisioned to unify First-Order Logic with nonmonotonic negation. We are currently looking into different ways of combining these. A possible direction is to use a nonmonotonic formalism such as circumscription [McCarthy, 1986].

Table 2.1 gives an overview of the features included in each of the language variants.

Feature	Core	DL	Flight	Rule	Full
Classical Negation (**neg**)	-	X	-	-	X
Existential Quantification	-	X	-	-	X
Disjunction	-	X	-	-	X
Meta Modeling	-	-	X	X	X
Default Negation (**naf**)	-	-	X	X	X
LP implication	-	-	X	X	X
Integrity Constraints	-	-	X	X	X
Function Symbols	-	-	-	X	X
Unsafe Rules	-	-	-	X	X

Table 2.1: WSML Variants and Feature Matrix

In the remainder of this deliverable we will use these requirements on the reasoner for the different WSML variants to evaluate the suitability of different reasoners for use with WSML.

[2] http://owl.man.ac.uk/factplusplus/
[3] http://www.racer-systems.com/
[4] http://www.mindswap.org/2003/pellet/index.shtml

3 Survey of Reasoners

This chapter presents the survey on existing reasoning implementations. Each reasoner implementation is described along the reasoning requirements for each of the WSML variants, which were outlined in chapter 2.

We describe reasoners from the areas of Logic Programming, First-order theorem proving and Description Logic reasoning implementations.

3.1 Logic Programming

In this section we describe a number of Logic Programming implementations. We are interested which of the WSML variants these implementations are able to deal with and to what extent, i.e. what kind of inferences they support. As one may expect, the reasoners from this category deal very well with query answering or instance retrieval for the WSML variants that are included in Logic Programming, namely WSML Core and WSML Flight, but they are not meant to be used as subsumption reasoners. Still, as shown in [Grosof et al., 2003], subsumption reasoning can be reduced to query answering for the subset of DL that intersects LP, named DLP (Description Logic Programs) on which WSML Core is based. A mapping function is used for translating from DL to LP, afterwards different types of DL queries being reduced to LP queries. Thus, LP reasoners can be used for performing subsumption with WSML Core.

3.1.1 SWI-Prolog

Prolog is a logical programming language based on first-order predicate calculus, restricted to allow only Horn clauses. The execution of a Prolog program is an application of theorem proving by first-order resolution. SWI-Prolog[1] is a free Prolog environment [Wielemaker, 2003], licensed under the Lesser GNU Public License.

As an add-on, SWI-Prolog offers support for storing and querying Semantic Web documents. It has a library named SWI-Prolog/XPCE Semantic Web Library that offers packages for reading, querying, and storing RDF, RDFS, and OWL, as well as XPCE libraries that provide visualization and editing for such documents. However, the support for OWL, offered as part of the Triple20 editor, is limited for now.

Another useful package of SWI-Prolog that can be used for developing hybrid Prolog/DL reasoners was implemented as part of the SEKT project[2]. The package contains a DIG description logic interface extension for high-level programming languages[**?**]. With the help of the package one can create a Prolog-based extended DL server that can call a standard DL reasoner that supports the DIG interface to provide regular DL services to client applications or that can act as an extended DL reasoner.

WSML-Core Since WSML-Core falls inside Horn Logic, SWI-Prolog has no problems regarding query answering in this language.

[1] http://www.swi-prolog.org
[2] http://www.sekt-project.com/

Subsumption reasoning within WSMO can also be handled by SWI-Prolog in the manner described in the introduction to this section. Other ways to perform subsumption reasoning could be using the SWI-Prolog/XPCE Semantic Web Library or using the extended DIG description logic interface. From the latter two approaches, the first approach would be handy for reasoning with WSML theories that are represented in RDF/OWL syntax (such exchange syntaxes are defined for WSML[3]), while the second implies using an external DL reasoner (such reasoners are further discussed in this survey).

SWI-Prolog can handle integrity constraints not directly, but by means of extra queries. It has limited datatype support, like any Prolog implementation. One can better talk about lexical elements instead of data-types in Prolog. The most general lexical element is the *term*. Special types of terms are *variables*, *numbers* (*integers or floats*), *atoms*, *strings*, *lists*. SWI-Prolog has a comprehensive set of built-in predicates.

The symbols *true* and *false* are present in SWI-Prolog in the form of the control predicates $true\backslash 0$ and $fail\backslash 0$.

In conclusion SWI-Prolog can be used as a reasoner for WSML-Core.

WSML-Flight SWI-Prolog has support for negation as failure and has a built-in predicate for equality, so it covers the minimal requirements for a WSML-Flight reasoner. However, meta-modelling is not directly supported.

WSML-Rule SWI-Prolog supports function symbols and unsafe rules, but it doesn't implement the well-founded semantics and thus, it does not deal with unstratified negation under this semantics.

WSML-DL SWI-Prolog by itself cannot handle the features of DL that are not in the intersection of DL with Horn Logic Programs. However, by using the extended DIG interface it can call an external DL reasoner for performing DL reasoning.

WSML-Full SWI-Prolog is not able to handle features added by WSML-Full compared with WSML-Flight.

Among the advantages of SWI-Prolog are that it is portable to many platforms, including almost all Unix/Linux platforms, Windows (95/98/ME and NT/2000/XP), MacOS X (using X11 for graphics) and many more. Binary distributions for most popular platforms (Windows, Linux and MacOS X) are regularly released. Also, the full source packages can be accessed through the CVS. One of its advantages over other LP reasoning engines is that it offers fast and flexible libraries for parsing SGML (HTML) and XML, RDF, store and query RDF, RDFS, and OWL, and an extended DIG interface which can be used for performing DL reasoning by calling external DL reasoners. As a disadvantage it can be mentioned the fact that it does not deal with non-stratified negation (compared with XSB Prolog, for example).

3.1.2 XSB

XSB[4] is a Logic Programming and Deductive Database system with support for different kinds of negation such as stratified negation and negation under the

[3] http://www.wsmo.org/TR/d16/d16.1/v0.3/
[4] http://xsb.sourceforge.org/

well-founded semantics. It also provides packages for evaluating F-Logic (see the FLORA-2 section) or a HiLog implementation.

The developers of XSB regard it as beyond Prolog because of the availability of SLG resolution [Chen and Warren, 1996] and the introduction of HiLog terms. SLG resolution enables the resolution of recursive queries that SLD resolution cannot deal with, and it also enables the use of well-founded semantics for non-stratified negation.

XSB also offers interfaces to other software systems, such as C, Java, Perl, ODBC, SModels, and Oracle. These interfaces also allow easy integration of new Built-in predicates.

WSML-Core XSB can be used for query answering in WSML-Core. However, integrity constraints (both value and cardinality constraints) are not directly supported but can be externally simulated by queries.

Regarding subsumption reasoning, XSB is not intended for this reasoning task. However, it can be handled in the manner described at the beginning of this section.

XSB allows the use of integers, strings and floating point numbers, and a set of built-in datatype predicates.

Consequently, XSB is a candidate reasoner WSML-Core.

WSML-Flight XSB implements equality and stratified negation. The support for HiLog also enables meta-modelling. Therefore, XSB can be used as a WSML-Flight reasoner.

WSML-Rule The features added in WSML-Rule, namely function symbols and the use of unsafe rules, are supported by XSB. Furthermore, XSB supports negation under the Well-Founded Semantics. Therefore, XSB is seen as candidate reasoner for WSML-Rule.

WSML-DL XSB cannot handle disjunction on the head and classical negation. Therefore, it can not be used as a reasoner for WSML-DL.

WSML-Full As XSB doesn't handle the proper subset WSML-DL, it will not be possible to use it as a reasoner for WSML-Full.

XSB is available for UNIX and Windows systems, and it is openly distributed under LGPL license. Its major disadvantage is its limited support for subsumption reasoning. In summary, XSB is seen as a suitable reasoner for WSML-Core, WSML-Flight and WSML-Rule, but not for WSML-DL and WSML-Full.

3.1.3 $\mathcal{FLORA}$-2

$\mathcal{FLORA}$-2[5] is a rule-based knowledge representation formalism and a reasoner for this formalism. It is based on F-Logic [Kifer et al., 1995], HiLog [Chen et al., 1993], and Transaction Logic [Bonner and Kifer, 1998], unifying these languages in a formalism which inherits important features from each of them: object-oriented features from F-Logic, reification and meta-information processing capabilities from HiLog, and declarative programming of "procedural knowledge" from Transaction Logic.

[5] http://flora.sourceforge.net/

The underlying inference engine of the reasoner is XSB Prolog[6]. In fact, another view of $\mathcal{FLORA}$-2 is as syntactic sugar on top of XSB. [Balaban, 1995] argues that DL is a subset of F-Logic, thus from either perspective, as a unifying language for the three formalisms or as syntactic sugar on top of XSB, $\mathcal{FLORA}$-2 is more than Horn Logic. An OWL reasoner was built on top of $\mathcal{FLORA}$-2[**?**] which offers support for OWL Full. The F-OWL approach is neither decidable nor complete. The reasoner called F-OWL[7] is built around a set of rules that reasons over the data model of RDF/RDF-S and OWL, a set of rules that maps XMLDataTypes into XSB terms, a set of rules that performs ontology consistency checks, and a set of rules that provides an interface between the upper Java API calls to the lower layer Flora-2/XSB rules. Although F-OWL is a stand-alone system we regard it in this section as an extension of Flora-2 that can be used for DL reasoning.

More details about $\mathcal{FLORA}$-2 can be found in the $\mathcal{FLORA}$-2 manual[8].

WSML-Core $\mathcal{FLORA}$-2 is well-suited for query answering in WSML-Core; it implements a superset of Datalog, and it provides built-in predicates for strings and integers, among others. Regarding integrity constraints, they are not directly supported by $\mathcal{FLORA}$-2, as in pure Prolog based systems such as XSB. They can be handled by defining rules with the integrity condition on the body and a special symbol e.g. *false* on the head. For checking the integrity of a knowledge base $\mathcal{KB}$, it has to be checked that such special symbol is not entailed by $\mathcal{KB}$ i.e. $\mathcal{KB} \not\models false$. However, notice that the semantics of integrity constraints is defined outside the reasoner. Therefore, $\mathcal{FLORA}$-2 fulfills the requirements for query answering in WSML-Core if integrity constraints are handled in the way discussed above.

$\mathcal{FLORA}$-2 is not intended as a DL reasoner, but subsumption reasoning can be done in the manner described in the beginning of this section. F-OWL can be used for providing subsumption reasoning, as well. This would be useful for WSML-Core specifications which use the OWL syntax.

Thus, $\mathcal{FLORA}$-2 fulfills all the requirements for a WSML-Core reasoner.

WSML-Flight $\mathcal{FLORA}$-2 implements equality and stratified negation. Meta-modelling is also built into the reasoner. Therefore, it can be used as a WSML-Flight reasoner.

WSML-Rule Function symbols and unsafe rules are supported by $\mathcal{FLORA}$-2. Furthermore, $\mathcal{FLORA}$-2 supports the Well-Founded Semantics. Therefore, $\mathcal{FLORA}$-2 is seen as a candidate reasoner for WSML-Rule.

WSML-DL $\mathcal{FLORA}$-2 by itself cannot handle DL reasoning. However, F-OWL could be used as a reasoner for WSML-DL.

WSML-Full $\mathcal{FLORA}$-2 provides a good support for the extensions added on top of WSML-Core in the direction of Logic Programming. F-OWL offers support for DL reasoning Full. However, FOL is not included in the union of LP and DL. In addition, $\mathcal{FLORA}$-2 it uses minimal model semantics, not first-order semantics. Thus, it is not clear whether it can be used as a reasoner for WSML-Full.

[6] http://xsb.sourceforge.net/
[7] fowl.sourceforge.net/about.html
[8] http://flora.sourceforge.net/docs/floramanual.pdf

The main advantages of $\mathcal{FLORA}$-2 are that it is distributed under LGPL license, it is constantly been improved i.e. it is "alive", and it is available for both Windows and POSIX operating systems.

In addition, it has support for HiLog and transaction logic, and a Java API is already available[9].

$\mathcal{FLORA}$-2 is fully based on XSB Prolog and relies on the tabling of XSB. We are not aware of any ports planned to other Prolog systems.

The major disadvantage is that $\mathcal{FLORA}$-2 is not intended as a DL reasoner and, therefore, it is hard to be efficiently used for subsumption reasoning in WSML-Core and WSML-DL.

3.1.4 TRIPLE

TRIPLE[10] is the name of both: a language for the Semantic Web and the corresponding reasoning system. The language TRIPLE is a layered and modular language for the Semantic Web (especially for the querying and transformation of RDF models) which bases on Horn Logic extended by F-Logic features and RDF constructs. Because these extensions are only syntactical, the language can be translated to Prolog and processed by a Prolog system. Correspondingly, the TRIPLE inferencing engine bases on the XSB Prolog system which has been described in section 3.1.2.

WSML-Core TRIPLE can be used for query answering in WSML-Core. For the purpose of subsumption reasoning, TRIPLE is suitable to only a limited extent. Subsumption reasoning can be handled by TRIPLE or XSB, respectively, only by the approach of Grosof et al. which is mentioned at the beginning of this section. I.e. the logic programming syntax of WSML-Core has to be t Triple allows the usage of the same data type predicates like XSB (cf. section 3.1.2).

This means that TRIPLE can be used for query answering, while it is limited regarding subsumption reasoning.

WSML-Flight TRIPLE implements equality and well-founded negation. However, as the TRIPLE system uses XSB and maps the negation operator to a negation operator of XSB, another negation operator of XSB could easily be used in this mapping. XSB implements stratified negation and thus it would be possible to use TRIPLE with stratified negation as well. Meta-modelling is also possible with TRIPLE. Therefore, some adjustments would enable its usage as a WSML Flight reasoner.

WSML-Rule TRIPLE supports function symbols, unsafe rules and negation under the well-founded semantics. Therefore WSML-Rule can be implemented on top of TRIPLE.

WSML-DL TRIPLE cannot handle disjunction in the head and is even limited for reasoning with the proper subset WSML-Core. Therefore, we do not see TRIPLE as a candidate for a WSML-DL reasoner.

[9] http://www.ontotext.com/downloads/

[10] http://triple.semanticweb.org/

WSML-Full TRIPLE cannot handle WSML-Full, as it cannot handle its proper subset WSML-DL.

The main advantage of TRIPLE is that it is distributed under BSD License. However, the newest version of TRIPLE is available only for Unix/Linux systems. The limitation to Linux might become inconvenient, when we want to combine several reasoners for the different WSML variants. The main disadvantage of TRIPLE is that it is not developed further at the moment.

3.1.5 OntoBroker

Ontobroker is a reasoner for reasoning with ontologies formalized in F-Logic or in Datalog/Prolog. It can be described as a main memory deductive, object oriented database system. The Ontobroker system is developed and distributed by the company Ontoprise[11].

WSML-Core Ontobroker is not a DL reasoning engine. Furthermore, it also doesn't implement disjunctive Datalog. Therefore, subsumption checking cannot be performed with Ontobroker.
Ontobroker implements a Datalog reasoning engine and can handle both integrity constraints and a huge amount of built-in predicates. Hence, it fulfills all the requirements for performing the query answering task within ontologies formulated in WSML-Core.

WSML-Flight Because Ontobroker implements both the equality symbol and stratified negation, all requirements for reasoning with ontologies in WSML-Flight are fulfilled.
Furthermore, Ontobroker has native support for meta-modeling.

WSML-Rule OntoBroker supports function symbols, as well as Well-Founded Semantics, as well as unsafe rules. Therefore, WSML-Rule could be implemented on top of OntoBroker.

WSML-DL Since Ontobroker is neither a DL reasoner nor a full First Order or a modal logic reasoner, it cannot deal with the most part of WSML-DL.

WSML-Full Ontobroker is not able to handle features added by WSML-Full compared with WSML-Flight as it cannot deal with the Description Logic subset.

A major drawback of Ontobroker is that it is being developed within a company and isn't available freely for anyone to use.

3.1.5.1 SILRI and MINS

A sidenote is in place on one of Ontobroker's predecessors. SILRI [12] is a less efficient predecessor version of Ontobroker with less features, which has been made public by Ontoprise.

DERI is currently starting efforts towards the development of a rule engine tailored for WSML, called MINS, which essentially is or rather will be a reimplementation of the SILRI rule system. Currently, it is intended to being adjusted

[11] ttp://www.ontoprise.de/home_en
[12] http://ontobroker.semanticweb.org/silri/

for the purpose of reasoning with ontologies written in WSML-Core, WSML-Flight and WSML-Rule. The source code and an online demo can be found on the MINS web site[13].

3.1.6 DLV

The DLV system [14] is being developed for several years as joint work of the University of Calabria and Vienna University of Technology.

DLV and the SMODELS system discussed in the subsequent section distinguish from the above-mentioned Prolog-Based systems in that these systems implement fully declarative logic programming in a purer sense than PROLOG-like systems. These systems compute models rather than answering queries in a top-down fashion.

The DLV system is an efficient engine for computing answer sets (i.e. stable models for logic programs with negation under the extension with classical negation and disjunction [Gelfond and Lifschitz, 1991]. DLV uses as core input language safe Datalog programs (cf. [Ullman, 1989]) with disjunction in rule heads and default negation in rule bodies. Programs are safe if every variable occurring in head literals or default negated body literals also occurs in at least one non-default-negated body literal. Note that in DLV head and body literals may be classically negated. A logic Program Π is *safe* if all of its rules are safe.

Note that the safety restriction is only syntactical but does not really affect the expressive power of the language in any way.

DLV computes answer sets, i.e. it follows the stable model semantics but not the well-founded semantics.

DLV provides some preliminary APIs such as a wrapper for using DLV from Java [15] and ongoing work on an ODBC interface connecting to relational databases. This interface has been considerably improved in the latest release.

Built-In Predicates and Aggregates With respect to the built-ins prescribed in WSML, built-in support in DLV is currently limited as follows.

DLV allows in its current release for a limited set of the built-in predicates "$A < B$", "$A <= B$", "$A > B$", "$A >= B$", "$A! = B$" with the obvious meaning of less-than, less-or-equal, greater-than, greater-or-equal and inequality for strings and numbers which can be used in the positive bodies of DLV rules and constraints.

DLV currently does not support full arithmetics but supports some built-in predicates, which can be used to "emulate" range restricted integer arithmetics: The arithmetic built-ins "$A = B + C$" and "$A = B * C$" which stand for integer addition and multiplication, and the predicate "$\#int(X)$" which enumerates all integers (up to a user-defined limit).

The restrictions on arithmetic built-ins will be lifted in one of the next releases where integration of a rich library of built-ins and also an API for users to add further functional built-ins are planned.

Furthermore, DLV also supports a front-end to interface the DL-Reasoner racer, as described in [Eiter et al., 2004].[16] Support to interface with other reasoners will probably be extended and generalized in future versions. Syntactical extension towards higher-order syntax [Eiter et al., 2005] are currently under discussion, but not yet part of the current release version.

[13] http://dev1.deri.at/mins/

[14] ⟨URL:http://www.dlvsystem.com⟩

[15] http://160.97.47.246:8080/wrapper/

[16] http://www.kr.tuwien.ac.at/staff/roman/semweblp/

WSML-Core Since WSML-Core (without nominals) falls in the Datalog subset of Horn-Logic, DLV can serve for several reasoning tasks. As the other Logic programming based engines, DLV is more suitable for query answering than for subsumption reasoning. However, to the expressiveness of Logic Programming under the stable model semantics, restricted forms of subsumption, namely conjunctive query containment which is well known to be NP-complete [Chandra and Merlin, 1977], could be checked, after Logspace transformations. As for datatype support the current basic support for built-ins and arithmetics seems not yet sufficient. APIs for extending the set of available built-ins is currently not provided.

Integrity constraints are naturally supported inside the stable model semantics. This distinguishes engines for logic programs under the stable model from other semantics for Logic Programming where Integrity constraints can only be emulated by additional queries.

Due to the safety restriction, the *true*-keyword in WSML would need to be emulated by a unary relation HU which contains the whole Herbrand Universe, for rules only having true in the body.

WSML-Flight Stratified Negation, more precisely stratified default negation is naturally supported in the language of DLV. Moreover, DLV allows the use of classical negation, which can be viewed as syntactic sugar in answer set semantics compared with stable model semantics. DLV supports inequality in the body. However, meta-modeling is not directly supported in DLV.

Thus, DLV fulfills all requirements for reasoning with WSML-Flight.

WSML-Rule DLV does not support function symbols or unsafe rules. Furthermore, DLV supports the stable model semantics which defers from the well-founded semantics which WSML-Rule is based upon for non-stratified programs. Thus, the restrictions we would have to apply to WSML-Rule to make use of the DLV reasoner make the language equivalent to WSML-Flight. Therefore, we do not consider DLV as a reasoner for WSML-Rule as is. Extensions towards reasoning for (limited) classes of programs with function symbols similar to what is mentioned in the literature [Bonatti, 2004] are currently being discussed and will be considered in future releases.

WSML-DL DLV is on the one hand not tailored for reasoning about description logics. On the other hand the translations of DL to disjunctive Datalog outlined in [Hustadt et al., 2004b] can be used instantly. Note that the full expressiveness of the language supported by DLV is not needed in this translation, since it only uses positive disjunctive Datalog without negation in the body.

WSML-Full DLV is not able to handle features added by WSML-Full compared with WSML-Flight. It is even not able to handle WSML-DL (except via interfacing to an external DL reasoner as pointed out above).

DLV is not being developed within an open-source license, and therefore not completely open for extensions. However, since it is developed in an academic environment, collaborations for further academic development are possible. Planned extensions of the system seem very promising for adoption towards reasoning in the context of WSML.

3.1.7 SMODELS and GNT

SMODELS [Niemelä, 1999; Simons et al., 2002][17] allows for the computation of stable models and well-founded models for normal logic programs, that is for Datalog programs with non-disjunctive heads. However, there is an extended prototype version for the evaluation of disjunctive logic programs as well, called GNT [Janhunen et al., 2000][18].

Syntactically, SMODELS imposes an even stronger restriction than rule safety in DLV by demanding that any variable in a rule is bounded to a so-called domain predicate in the rule body which is, intuitively, a predicate which is defined only positively (cf. [Niemelä, 1999] for details). Again, this restriction does not affect the expressive power of the language itself, but in some cases the weaker safety restriction of DLV allows for more concise problem encodings.

Arithmetics and Built-in Predicates Similar to DLV, SMODELS allows for a restricted form of integer arithmetics and lexicographic comparison predicates.

WSML As for the implementability for the various WSML variants, similar considerations apply as for DLV mentioned above. Differences show up in the fact that besides computation of stable models SMODELS also supports computation of well-founded models.

SMODELS is developed under the GNU Public Licence and provides more API support than DLV, also from third parties. For instance, also XSB mentioned above provides a direct interface to smodels called XASP enabling the use of stable model semantics from XSB.

3.1.8 KAON2

KOAN2[19] provides a hybrid reasoner which is able to reason with a large subset of OWL DL, corresponding to the Description Logic $\mathcal{SHIQ}(\mathbf{D})$ [Horrocks et al., 2000], and Disjunctive Datalog with stratified negation, along with basic built-in predicates to deal with integers and strings. The theoretical work underlying KAON2 can be found in [Hustadt et al., 2004a].

WSML-Core KAON2 can perform subsumption reasoning in the subset of WSML-Core which falls inside the $\mathcal{SHIQ}$ Description Logic.

Since KAON2 implements a Datalog engine and is able to handle integrity constraints, as well as rudimentary built-in predicates, it fulfills all the requirements for performing the query answering task efficiently with WSML-Core.

WSML-Flight Because KAON2 implements the equality symbol and stratified negation, all requirements for reasoning with WSML-Flight are fulfilled. KAON2 has been extended with additional meta-level rules in order to be able to handle meta-modeling. Consequently, KAON2 is a candidate reasoner for WSML-Flight.

WSML-Rule KAON2 does not support function symbols or unsafe rules. Furthermore, it does not support the Well-Founded Semantics. Therefore, we do not consider KAON2 as a reasoner for WSML-Rule.

[17] http://www.tcs.hut.fi/Software/smodels/
[18] http://www.tcs.hut.fi/Software/gnt/
[19] http://kaon2.semanticweb.org/

WSML-DL Since KAON2 is a DL reasoner for the $SHIQ(\mathbf{D})$ DL, it can deal all of WSML-DL.

WSML-Full KAON2 is not able to handle features added by WSML-Full compared with WSML-DL and WSML-Flight. However, it does allow for an interesting combination of Description Logics and rules, since Description Logic ontologies are reduced to disjunctive logic programs by KAON2 in order to allow for efficient query answering.

A major drawback of KAON2 is that it cannot deal with unstratified negation under the well-founded semantics. Furthermore, there is no support for built-in predicates yet.

The major advantage of KAON2 is that it is a very efficient reasoner when it comes to reasoning with Description Logics ontologies containing very large ABoxes and small TBoxes[Motik and Sattler, 2005]. KAON2 is owned by the company Ontoprise and is available only for non-commercial settings.

3.1.9 Summary

The surveyed Logic Programming implementations all agree on Datalog with stratified negation. This makes implementation of WSML-Flight on any of the surveyed reasoners possible.

The reasoners XSB, FLORA-2, and OntoBroker all support function symbols and support unstratified negation under the well-founded semantics. Thus, they are suitable for implementation of WSML-Rule.

Frame-based modeling is supported by FLORA-2 and OntoBroker. Native support for frame-based modeling makes the implementation of a WSML-Rule reasoner easier. Hence, FLORA-2 and OntoBroker are the preferred candidates for the implementation of WSML-Rule.

3.2 First-order theorem provers

First-order Logics are quite expressive logics which can be applied for reasoning in a wide range of domains. They provide a flexible general-purpose framework for representing domain knowledge and inferring new knowledge from a given knowledge base. Their theoretical properties are well-understood and the construction of deduction systems for First-order Logics has been a subject of investigation for several decades now [Davis, 2001]. Of course, the flexibility and expressiveness of this family of logics comes at a cost: Reasoning with these languages in general is undecidable[20] That means, when using these logics one looses in general a certain nice guarantee, namely to get an answer for *any* concrete input within a finite period of time when performing reasoning.

For this family of logics numerous calculi for checking logical entailment have been proposed and implemented. We have selected two of them that are discussed in the following sections. The reason underlying our choice is that both systems seem to be able to deal with larger knowledge bases during

[20]Depending on the concrete reasoning task the problem can be recursively enumerable or even not recursively enumerable. For instance, checking logical entailment of a formula from a knowledge base is semi-decidable since there are complete calculi that formalize logical entailment for First-order Logic. On the other hand, the set of satisfiable First-order Logic formulae is not recursively enumerable and thus checking satisfiability of arbitrary formulae in these logics is in general out of scope for algorithmic treatment.

reasoning which is a key property in our applications as well as one of the major problems for reasoners for these logics. Both systems at least have been applied in applications which are similar to the ones that we aim at.

3.2.1 Vampire

The Vampire system has been developed by Alexandre Riazanov and Andrei Voronkov at the University of Manchester, England. Several versions of the system have been tested and evaluated in the annual contest for automated theorem provers called CASC[21] and during the last years, the system won in the categories for First-order Logics with equality (FOF and MIX). It can be considered as one of the most powerful systems for fully-automated theorem proving in First-order Logics. In the following we refer to the most recent version of the system, namely Vampire 7.0 which participated in CASC in 2004.

One of the most interesting and distinguishing characteristics of this system is that the system designer spent significant effort in the development and design of powerful techniques for *implementing* theorem proving systems i.e. the overall technical infrastructure needed during proof search, such as indexing techniques for terms and formulae, runtime-compilation of code, careful memory-management, resource-oriented (or resource-aware) algorithms, etc. In this respect, there is currently no comparable system available.

Technical Overview. Vampire[22] [Riazanov and Voronkov, 2002] is an automatic theorem prover for first-order classical logic. Its kernel implements the calculi of ordered binary resolution and superposition for handling equality. The splitting rule and negative equality splitting are simulated by the introduction of new predicate definitions and dynamic folding of such definitions. A number of standard redundancy criteria and simplification techniques are used for pruning the search space: subsumption, tautology deletion (optionally modulo commutativity), subsumption resolution, rewriting by ordered unit equalities, basicness restrictions and irreducibility of substitution terms. The reduction orderings used are the standard Knuth-Bendix ordering and a special non-recursive version of the Knuth-Bendix ordering. A number of efficient indexing techniques is used to implement all major operations on sets of terms and clauses. Runtime algorithm specialization is used to accelerate some costly operations, e.g., checks of ordering constraints. Although the kernel of the system works only with clausal normal forms, the preprocessor component accepts a problem in the full first-order logic syntax, clausifies it and performs a number of useful transformations before passing the result to the kernel. When a theorem is proven, the system produces a verifiable proof, which validates both the clausification phase and the refutation of the CNF. The current release features a built-in proof checker for the clausifying phase, which will be extended to check complete proofs. Vampire 7.0 is implemented in C++.

Strategies. The Vampire kernel provides a fairly large number of features for strategy selection. The most important ones are:

- Choice of the main saturation procedure : (i) OTTER loop, with or without the Limited Resource Strategy, (ii) DISCOUNT loop. A variety of optional simplifications.
- Parameterized reduction orderings.

[21] `http://www.cs.miami.edu/~{}tptp/CASC/`
[22] `http://www.cs.man.ac.uk/~riazanov/Vampire`

- A number of built-in literal selection functions and different modes of comparing literals.
- Age-weight ratio that specifies how strongly lighter clauses are preferred for inference selection.
- Set-of-support strategy.

The automatic mode of Vampire 7.0 is derived from extensive experimental data obtained on problems from TPTP v2.6.0 problem library[23]. Input problems are classified taking into account simple syntactic properties, such as being Horn or non-Horn, presence of equality, etc. Additionally, the presence of some important kinds of axioms, such as set theory axioms, associativity and commutativity is taken into account. Every class of problems is assigned a fixed schedule consisting of a number of kernel strategies called one by one with different time limits.

WSML-Core Since WSML-Core without datatype predicates can be considered as a subset of the $\mathcal{SHOIN}$ Description Logic, which can be translated into First-order Logic [Hustadt et al., 2004a]. Thus, Vampire (as any First-order Logic theorem prover) can be used for reasoning with WSML-Core without datatype predicates. In fact, Vampire has already been evaluated in this respect and compared with the Description Logic inference system FACT[24] [Tsarkov and Horrocks, 2003a] and its re-implementation FaCT++ [Tsarkov et al., 2004] (see also Section 3.3.1). The results of the evaluation show that Vampire is in general slower[25] than the FACT(++) system, which is not surprising since FaCT++ is specifically tailored towards a significantly simpler logic. On the other hand, Vampire actually proved to be able to solve many questions with acceptable response time.

A standard way for dealing with concrete datatypes in theorem proving is to use an explicit axiomatization of datatype: One gives a set of formulae that constrain the interpretation of the symbols used to denote datatype operators to their intended semantics. Nonetheless, this way of dealing with datatypes is not a very efficient one and thus should only be considered as a work-around. Since some examples for Vampire Problem files indicate, the system seems to support some built-in datatypes starting from Version 7.0, though we do not know yet, what datatypes are supported and to what extend they are supported. Currently, there is no documentation available. Nonetheless, it is not clear whether datatype predicates or oracles will be supported in the way that is needed for WSML-Core.

Although Vampire can be used to check logical entailment of arbitrary formulae and thus can basically be used for the required reasoning tasks for WSML-Core, there are no special techniques for a query answering task in the style of deductive databases. Thus, it is unlikely that Vampire can perform comparably well (wrt. deductive databases) on this task. In the worst case, one has to apply a naive enumeration and check algorithm for exhaustive query answering which obviously is not a good solution.

The symbols *true* and *false* are supported by WSML-Core as well. *What about Vampire?*

WSML-Flight Vampire supports equality and the *standard* first-order negation. No other form of negation is implemented in the Vampire system, since

[23] `http://www.cs.miami.edu/~tptp/`
[24] `http://www.cs.man.ac.uk/~horrocks/FaCT/`
[25] Sometimes significantly slower!

it deals with classical Predicate Logic with equality. It does not provide specific (built-in) support for Meta-Modelling since this is not part of First-order Logic, though is can deal with Meta-Modelling as long as it can be encoded in First-order Logic.

WSML-Rule Function symbols are supported by Vampire though it does not implement variants of the negation symbol other than the standard First-order negation symbol. In particular, Vampire (as every First-order logic theorem prover) sticks to the open-world semantics. Thus, it can not be used for reasoning with descriptions in WSML-Rule.

WSML-DL Here, the same arguments as for WSML-Core apply: Vampire can basically be used for both envisioned reasoning tasks, nonetheless for query answering we expect the system to not perform well in comparison to special-purpose systems like deductive databases.

WSML-Full Since Vampire is a deduction system for classical Predicate Logic, it does not provide any support for non-monotonic reasoning. Thus, it can not be applied for reasoning with WSML-Full in general. However, it can be applied for the subset of WSML-Full which does not use non-monotonic features. In this subset, Vampire is likely to outperform most other systems when checking logical entailment. The same drawbacks as before apply for query answering, although for languages of this expressiveness we are not aware of any system which is specifically tailored towards the query answering task.

Further Remarks on Vampire. Vampire is under continuous development and evaluation. There are no indications that this will change in the near future. At present, there is no Handbook or detailed documentation for users available.

Vampire supports the TPTP format[26] as a standard-like input format for problem specifications. When vampire is started on such an input file, it tries to construct a proof for the conjecture included in the problem specification. The output of the prover indicates whether a proof has been found (actually the proof can be output as well), no proof could be constructed either within a given time limit or if such a proof is not possible at all[27].

It seems that Vampire currently does not really support some sort of knowledge base mode where one feeds a knowledge base into the prover and just invokes the proof search for various queries. Instead, one has to feed everything which is relevant for proving into the prover with every query. Clearly, this might be a drawback in many application in the Semantic Web since the domain knowledge is needed for all reasoning tasks whereby the domain knowledge is comparably stable wrt. the given conjectures to be checked (or queries to be answered). From private communication with the authors we know that such a feature could be supported by Vampire in the future if there are applications that need that feature.

3.2.2 SNARK

SNARK, SRI's New Automated Reasoning Kit, is a theorem prover intended for applications in artificial intelligence and software engineering. SNARK is geared toward dealing with large sets of assertions and thus it is of special

[26] `http://www.cs.miami.edu/~tptp/TPTP/TR/TPTPTR.shtml`

[27] In this case, one is really lucky since this the problematic case for FOL theorem provers, since non-termination of the proof search can happen here!

interest for us; it can be specialized with strategic controls that tune its performance; and it has facilities for integrating special-purpose reasoning procedures with general-purpose inference.

SNARK has been used as the reasoning component of SRI's High Performance Knowledge Base (HPKB) system[28] [Pease et al., 2000], which deduces answers to questions based on large repositories of information. It constitutes the deductive core of the NASA Amphion system[29], which composes software from components to meet users' specifications, e. g., to perform computations in planetary astronomy. SNARK has also been connected to Kestrel's SPECWARE environment [30] for software development.

Technical Overview. SNARK is a resolution-and-paramodulation theorem prover for first-order logic with equality. SNARK has provisions for both clausal and nonclausal reasoning, and it has an optional sort mechanism. It incorporates associative/commutative unification and a built-in decision procedure for reasoning about temporal points and intervals. Furthermore, it allows to use special-purpose external procedures (so-called Procedural Attachments) during the inference process and supports question answering applications to some extend. As input format for the prover, SNARK supports KIF+C (KIF plus Classes), the facility that enables SNARK to understand assertions and queries phrased in KIF [Genesereth, 1991; Genesereth and Fikes, 1992], the Knowledge Interchange Format, with some extensions from OKBC (Open Knowledge-Base Connectivity) [Chaudhri et al., 1998], an object-oriented framework for representing knowledge. This dialect was used in the HPKB project and other knowledge representation efforts. It has no special facilities for proof by mathematical induction. It has some capabilities for abductive reasoning, which have been used in natural-language applications. SNARK is implemented in an easily portable subset of COMMON LISP.

SNARK is a refutation system; in other words, rather that trying to show directly that some assertions imply a desired conclusion, it attempts to show that the assertions and the negation of the conclusion imply a contradiction. It is an agenda-based system; that is, in seeking a refutation, it will put the assertions and the negation of the conclusion on an agenda. An agenda is a list of formulas. When a formula reaches the top of the agenda, SNARK will perform selected inferences involving that formula and the previously processed formulas. The consequences of those inferences are added to the agenda. This process continues until the propositional symbol false is derived; this means that a contradiction has been deduced and the refutation is complete. The user has considerable control over the position at which a newly derived formula is placed on the agenda; this is one way in which a knowledgeable user can tailor SNARK's search strategy to a particular application.

WSML-Core SNARK can be used for reasoning with WSML-Core without datatype predicates. Via procedural attachments it seems to be possible to add special-purpose decision procedures for concrete datatypes and datatype predicates. Thus, extending SNARK to support full WSML-Core should be possible. SNARK was intended to support query answering applications as well and thus seems to support the derivation from multiple concrete answers to queries of a knowledge base. At present, we do not have further details on this feature and thus it is unclear whether SNARK is significantly better than other FOL provers in this respect. However, at least there is a guarantee for

[28] http://www.ai.sri.com/project/HPKB
[29] http://ase.arc.nasa.gov/docs/amphion.html
[30] http://www.kestrel.edu/HTML/prototypes/specware.html

minimal support for the query answering reasoning task. An interesting feature of SNARK is that it provides support for sorts. Since it uses an extension of KIF by classes as a possible input format it as well provides some rudimentary support for ontology reasoning. Whether reasoning in the presence of sorts in problem specifications is more efficient than with provers that do not support sorts still has to be evaluated.

WSML-Flight SNARK supports equality but no non-monotonic form of negation. As in the case of Vampire it does not provide specific (built-in) support for Meta-Modelling since this is not part of First-order Logic, though is can deal with Meta-Modelling as long as it can be encoded in First-order Logic.

WSML-Rule The system supports function symbols though it does not implement non-monotonic variants of the negation symbol. Thus, it can not be used for reasoning with descriptions in WSML-Rule.

WSML-DL Here the same arguments as for WSML-Core apply: SNARK can basically be used for reasoning with WSML-DL descriptions. Actually, there is a First-order axiomatization of OWL (including OWL-Full) in the input format of SNARK available. The authors evaluated the axiomatization as well as reasoning with OWL using the SNARK deduction system [Waldinger, 2004].

WSML-Full Since SNARK is a deduction system for classical Predicate Logic, it does not provide any support for non-monotonic reasoning. Thus, it can not be applied for reasoning with WSML-Full in general. However, it can be applied for the subset of WSML-Full which does not use non-monotonic features.

3.2.3 Summary

First-order logic theorem provers are flexible and very general tools. Often they are too general for specific application and thus perhaps unnecessarily heavy-weight. One general difficulty when applying deduction systems for First-order Logic in some application is that most often one has the opportunity to set a lot of parameters to affect the proof search and thus the efficiency of theorem proving in a specific application at hand. This freedom at the same time can be a burden if the system does not provide a strong automatic mode where the parameters are determined automatically by syntactically analyzing the input data. Customizing a automated theorem prover for First-order Predicate Logic is a non-trivial task which is needed in order to get reasonable performance within specific applications. The configuration process requires a deep understanding of how a system works and what formulae one typically has to deal with in the specific application.

First-order logic theorem provers are thus bulky and can anyway just be used for WSML-Core and WSML-DL. We conclude that a proper Description Logic or SHIQ reasoner would be much more efficient and easier to use for both WSML-Core and WSML-DL than a full first-order reasoner.

3.3 Description Logic reasoners

We describe three state-of-the-art Description Logic reasoners, namely: FaCT++, RACER, and Pellet. Note that KAON2, earlier described in Section 3.1.8, is also a Description Logic reasoner. In fact, it is a Hybrid reasoner, able to reason with both Description Logic ontologies and Disjunctive Datalog programs.

3.3.1 FaCT++

FaCT++[31] is a DL reasoner for the $\mathcal{SHIOQ}(\mathcal{D})$ Description Logic. It is a C++ re-implementation of the DL reasoner FaCT [Horrocks, 1998] which has been implemented in Lisp. Appart from the enhancement of efficiency and portability caused by the change of the programming language, a different internal architecture, new optimizations and new features have been added. More details can be found in [Tsarkov and Horrocks, 2003b].

WSML-Core FaCT++ can handle $\mathcal{SHIQ}(\mathcal{D})$. Unfortunately, the data types supported by FaCT++ are just strings and integers. Therefore, its use as a WSML-Core reasoner is possible but still limited as there is no full data types support. FaCT++ can provide subsumption reasoning for WSML-Core.

As currently no A-Box reasoning is implemented within FaCT++, all kinds of query answering are completely impossible. The However, there are plans to extend FaCT++ by means for A-box reasoning by end of this year. However, A-Box reasoning in FaCT++ and Description Logics reasoners in general is not efficient.

WSML-Flight FaCT++ being a Description Logics reasoner does not support Datalog, default negation and meta-modelling. Therefore, it is not suitable as a reasoner for WSML-Flight.

WSML-Rule FaCT++ being a Description Logics reasoner does not support function symbols and it also does not support any kinds of nonmonotonic negation required by logic programming in general. This means, that unstratified negation under the Well-Founded Semantics required by WSML-Rule is not supported as well. Therefore, it is not suitable for reasoning with WSML-Rule.

WSML-DL FaCT++ can handle $\mathcal{SHIQ}(\mathcal{D})$. Consequently, it is appropriate as a WSML-DL reasoner. However, the lack of support for other data types than strings and integers and the lack of support for A-Box reasoning, are clear disadvantages when it comes to a choice between the available Description Logics reasoners.

WSML-Full FaCT++ does not support any kind of non monotonicity and therefore, it is not suitable as a reasoner for WSML-Full.

The main advantages of FaCT++ is that it provides support for the DIG interface which is a standardised XML interface. By means of DIG, a web interface can be realized. Furthermore, FaCT++ is a very efficient and optimized T-Box reasoning engine.

[31] http://owl.man.ac.uk/factplusplus/

FaCT++ is available for Windows, Linux and MAC operating systems. It is distributed under GPL license, i.e. every piece of software depending on it must also be open source, which might be an disadvantage in some cases.

3.3.2 RACER

RACER[32] is a DL reasoner for the $\mathcal{SHIQ}$ Description Logic. It also provides limited support for concrete domains.

WSML-Core As said before, RACER can handle $\mathcal{SHIQ}$. RACER can handle datatypes, namely natural, integer, real and complex numbers, and strings. Thus, RACER can provide subsumption reasoning for WSML-Core.

RACER provides support for ABox reasoning and is not fixed to the Unique Name Assumption (UNA). However, the UNA can be enabled globally. The ABox reasoning efficiency of Racer is worse when compared to Logic Programming reasoners when the ABoxes are large (cf. [Motik and Sattler, 2005]. Therefore, RACER can be considered as a reasoner for query answering in WSML-Core, but it is outperformed by KAON2 for large ABoxes.

WSML-Flight Racer provides an equality predicate for individuals. But it does not support neither default negation nor meta-modelling. Thus, it does not fulfill the requirements for reasoning with WSML-Flight.

WSML-Rule RACER does not provide support for function symbols. Furthermore, it also does nor support neither default negation nor negation under the well-founded semantics. Therefore, it is not suitable as a WSML-Rule reasoner.

WSML-DL RACER can handle $\mathcal{SHIQ}$ and can be used as a reasoner for WSML-DL with sound and complete reasoning for ontologies including ABoxes and TBoxes.

WSML-Full RACER does not support any non-monotonic features and it does not support WSML-Flight and WSML-Rule which are proper subsets of WSML-Full. Therefore, it can not be used for reasoning with WSML-Full.

The main advantage of RACER is that it can be considered as a state-of-the-art TBox DL reasoner. In [Motik and Sattler, 2005] it has been shown to perform faster TBox reasoning than KAON2 and Pellet. Therefore, it would be the reasoner of choice for TBox reasoning with WSML-DL. However, KAON2 provides the most efficient support for ABox reasoning with WSML-DL when the ontology consists of a large ABox and a small TBox.

The major disadvantage is that RACER is only provided free of charge for universities and research labs, and that the source code is not publicly available.

3.3.3 Pellet

Pellet[33] is a sound and complete DL reasoner for the $\mathcal{SHIN}\mathcal{Q}(\mathcal{D})$ and $\mathcal{SHON}\mathcal{Q}$ Description Logics, and provides by means of these sound and complete algorithms support for sound and incomplete reasoning with $\mathcal{SHOIN}(\mathcal{D})$.

[32] http://www.sts.tu-harburg.de/ r.f.moeller/racer/
[33] http://www.mindswap.org/2003/pellet/index.shtml

WSML-Core Pellet can reason with XML Schema data types. It also provides support for simple user-defined data types that are derived from XML Schema data types. It doesn't adhere to the Unique Name Assumption.

By being a $\mathcal{SHOIQ}$ reasoner, Pellet can be used for subsumption reasoning with WSML-Core. It also can be used for query answering with WSML-Core. [Motik and Sattler, 2005] showed that it is a competitive DL reasoner. However, KAON2 outperforms it for ontologies with large ABoxes and small TBoxes while Races seems to be more efficient regarding TBox reasoning.

WSML-Flight Pellet supports neither default negation nor meta-modelling. Thus, it does not fulfill all the requirements for a WSML-Flight reasoner.

WSML-Rule Pellet supports neither function symbols nor negation under the well-founded semantics. Therefore, it is not suitable as a WSML-Rule reasoner.

WSML-DL Pellet can be used as a reasoner for WSML-DL as it supports the SHIQ Description Logic.

WSML-Full Pellet does not support any non-monotonic features and it does not support WSML-Flight and WSML-Rule which are proper subsets of WSML-Full. Therefore, it can not be considered as a reasoner for WSML-Full.

The main advantage of Pellet is its support of XML Schema data types. The other DL reasoners discussed in this deliverable, have a much more restricted data type support. Another advantage is that it is written in Java and it is freely available. It supports the DIG interface like the other DL reasoners described in this deliverable and seems to be competitive to KAON2 and Racer.

3.3.4 Summary

The surveyed Description Logic reasoners all agree on the $\mathcal{SHIQ}(\mathbf{D})$ Description Logic, which underlies WSML-DL. Thus, all three reasoners could be used for WSML-DL.

However, FaCT++ doesn't provide support for ABox reasoning currently. This is an disadvantage in comparison to the other reasoners that support ABox reasoning. However, as written in the section that describes FaCT++ the delevopers of FaCT++ plan to release a new version of FaCT++ with support for ABox reasoning soon.

In the efficiency comparison of [Motik and Sattler, 2005], it has been shown that KAON2 is very efficient for ontologies with large ABoxes and small TBoxes and Racer is very efficient for TBox reasoning in general. However, FaCT++ has not been tested and it remains to be seen how FaCT++ performs for TBox reasoning. It also remains to be seen how efficient the new version of FaCT++ providing support for ABox reasoning as well will be.

4 Conclusions and Future Work

In this deliverable, we have surveyed the requirements on a reasoner for the WSML variants. Furthermore, several already existing reasoners have been surveyed and evaluated in the context of possibilities of usage for the different WSML variants.

4.1 Related Developments

The survey of reasoners applicable for reasoning has been closely related with the development of the WSMO discovery engine (deliverable D10 [Keller et al., 2005]), which has provided requirements on the WSML reasoners and which will eventually make use of WSML reasoners.

Other related developments are:

- WSMO4J[1], which provides a data model in Java for WSML and also provides (de-)serializers for the different WSML syntaxes except of WSML-DL and WSML-Full. WSMO4J can be extended to connect with the specific reasoners to be used for WSML.
- The WSML validator[2] currently provides validation services for the basic syntax defined in D2v1.0 [Roman et al., 2004]. We expect the validator to be extended to handle the different WSML variants under development in D16.1 [de Bruijn et al., 2005]. However, we expect that the WSML validator will eventually be subsumed by WSMO4J.
- The WSML reasoner implementation efforts consist of a reasoner infrastructure and wrapper implementations that enable the usage of existing core reasoners. As core reasoners currently FaCT++, $\mathcal{FLORA}$-2, DLV, Mandrake and MINS are used. An implementation of WSML-Rule with Ontobroker as core reasoner exists also, but is not developed further.
- The implementation efforts for the DERI logic programming reasoner MINS which bases on SiLRI will support the logic programming variants of WSML, i.e. WSML-Core, WSML-Flight and WSML-Rule.
- WSMX provides the reference implementation for WSMO. WSMX makes use of pluggable reasoning services. The only component in WSMX of which we are aware that it uses a reasoner is the mediation component, which uses the $\mathcal{FLORA}$-2 [Yang et al., 2003] logic programming engine. We expect that in future versions, WSMX will make use of the reasoners to be provided in the context of this deliverable for tasks such as mediation, discovery and composition. Therefore, future versions of this deliverable will also take specific requirements on the reasoners coming from WSMX into account.

[1] http://wsmo4j.sourceforge.net/
[2] http://dev1.deri.at:8080/wsml/validator.html

Acknowledgements

The work is funded by the European Commission under the projects DIP, Knowledge Web, InfraWebs, SEKT, SWWS, ASG and Esperonto; by Science Foundation Ireland under the DERI-Lion project; by the FIT-IT (Forschung, Innovation, Technologie - Informationstechnologie) under the projects RW^2 and TSC.

The editors would like to thank to all the members of the WSML working group for their advice and input into this document.

Bibliography

[Baader et al., 2003] Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., and Patel-Schneider, P. F., editors (2003). *The Description Logic Handbook*. Cambridge University Press.

[Balaban, 1995] Balaban, M. (1995). The f-logic approach for description languages. *Annals of Mathematics and Artificial Intelligence*, 15(1):19–60.

[Bonatti, 2004] Bonatti, P. A. (2004). Reasoning with infinite stable models. *Artificial Intelligence*, 156(1):75–111.

[Bonner and Kifer, 1998] Bonner, A. and Kifer, M. (1998). A logic for programming database transactions. In Chomicki, J. and Saake, G., editors, *Logics for Databases and Information Systems*, chapter 5, pages 117–166. Kluwer Academic Publishers.

[Calvanese et al., 1998] Calvanese, D., Giancomo, G. D., and Lenzerini, M. (1998). On the decidability of query containment under constraints. In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'98)*, pages 149–158.

[Calvanese et al., 2003] Calvanese, D., Giancomo, G. D., and Vardi, M. Y. (2003). Decidable containment of recursive queries. In *Proc. of the The 9th International Conference on Database Theory (ICDT 2003*, pages 327–342.

[Chandra and Merlin, 1977] Chandra, A. K. and Merlin, P. M. (1977). Optimal Implementation of Conjunctive Queries in Relational Data Bases. In *Conference Record of the Ninth Annual ACM Symposium on Theory of Computing*, pages 77–90.

[Chaudhri et al., 1998] Chaudhri, V. K., Farquhar, A., Fikes, R., Karp, P. D., and Rice, J. P. (1998). OKBC: A programmatic foundation for knowledge base interoperability. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pages 600–607, Madison, Wisconsin, USA. MIT Press.

[Chen et al., 1993] Chen, W., Kifer, M., and Warren, D. S. (1993). HILOG: A foundation for higher-order logic programming. *Journal of Logic Programming*, 15(3):187–230.

[Chen and Warren, 1996] Chen, W. and Warren, D. S. (1996). Tabled evaluation with delaying for general logic programs. *Journal of the ACM*, 43(1):20–74.

[Dantsin et al., 2001] Dantsin, E., Eiter, T., Gottlob, G., and Voronkov, A. (2001). Complexity and expressive power of logic programming. *ACM Computing Surveys (CSUR)*, 33(3):374–425.

[Davis, 2001] Davis, M. (2001). The early history of automated deduction. In Robinson, A. and Voronkov, A., editors, *Handbook of Automated Reasoning*, volume I, chapter 1, pages 3–15. Elsevier Science.

[de Bruijn et al., 2005] de Bruijn, J., Lausen, H., Krummenacher, R., Polleres, A., Predoiu, L., Kifer, M., and Fensel, D. (2005). The web service modeling language WSML. Technical report, WSML. WSML Final Draft D16.1v0.21. http://www.wsmo.org/TR/d16/d16.1/v0.21/.

[de Bruijn et al., 2004a] de Bruijn, J., Polleres, A., Lara, R., and Fensel, D. (2004a). OWL$^-$. Deliverable d20.1v0.2, WSML.
Available from http://www.wsmo.org/2004/d20/d20.1/v0.2/.

[de Bruijn et al., 2004b] de Bruijn, J., Polleres, A., Lara, R., and Fensel, D. (2004b). OWL flight. Deliverable d20.3v0.1, WSML.
Available from http://www.wsmo.org/2004/d20/d20.3/v0.1/.

[Eiter et al., 2005] Eiter, T., Ianni, G., Schindlauer, R., and Tompits, H. (2005). A uniform integration of higher-order reasoning and external evaluations in answer-set programming. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, Edinbutgh, Scotland.

[Eiter et al., 2004] Eiter, T., Lukasiewicz, T., Schindlauer, R., and Tompits, H. (2004). Combining answer set programming with description logics for the semantic web. In *Proc. of the International Conference of Knowledge Representation and Reasoning (KR04)*.

[Fensel et al., 2000] Fensel, D., Angele, J., Decker, S., Erdmann, M., Schnurr, H.-P., Studer, R., and Witt, A. (2000). Lessons learned from applying AI to the web. *International Journal of Cooperative Information Systems*, 9(4):361–382.

[Fitting, 1996] Fitting, M. (1996). *First-Order Logic and Automated Theorem Proving*. Springer-Verlag, second edition edition.

[Gelder et al., 1988] Gelder, A. V., Ross, K., and Schlipf, J. S. (1988). Unfounded sets and well-founded semantics for general logic programs. In *Proceedings 7th ACM Symposium on Principles of Database Systems*, pages 221–230, Austin, Texas. ACM.

[Gelfond and Lifschitz, 1991] Gelfond, M. and Lifschitz, V. (1991). Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3/4):365–386.

[Genesereth, 1991] Genesereth, M. (1991). Knowledge interchange format. In Allenet, J. et al., editors, *Proceedings of the Second International Conference on the Principles of Knowledge Representation and Reasoning (KR-91)*, pages 238–249. Morgan Kaufman. See also http://logic.stanford.edu/kif/kif.html.

[Genesereth and Fikes, 1992] Genesereth, M. R. and Fikes, R. E. (1992). Knowledge interchange format, version 3.0 reference manual. Technical report logic-92-1, Computer Science Department, Stanford University, Stanford, US,.

[Grosof et al., 2003] Grosof, B. N., Horrocks, I., Volz, R., and Decker, S. (2003). Description logic programs: Combining logic programs with description logic. In *Proc. Intl. Conf. on the World Wide Web (WWW-2003)*, Budapest, Hungary.

[Horrocks, 1998] Horrocks, I. (1998). The fact system. In de Swart, H., editor, *Automated Reasoning with Analytic Tableaux and Related Methods: International Conference Tableaux98*, number 1397 in Lecture Notes in Artificial Intelligence, pages 307–312. Springer-Verlag.

[Horrocks and Patel-Schneider, 2003] Horrocks, I. and Patel-Schneider, P. F. (2003). Reducing OWL entailment to description logic satisfiability. In *Proc. of the 2003 International Semantic Web Conference (ISWC 2003)*, Sanibel Island, Florida.

[Horrocks et al., 2000] Horrocks, I., Sattler, U., and Tobies, S. (2000). Practical reasoning for very expressive description logics. *Logic Journal of the IGPL*, 8(3):239–264.

[Hustadt et al., 2004a] Hustadt, U., Motik, B., and Sattler, U. (2004a). Reasoning for description logics around SHIQ in a resolution framework. Technical Report 3-8-04/04, FZI.

[Hustadt et al., 2004b] Hustadt, U., Motik, B., and Sattler, U. (2004b). Reducing $SHIQ^-$ description logic to disjunctive logic programs. FZI-Report 1-8-11/03, revised version, FZI.

[Janhunen et al., 2000] Janhunen, T., Niemelä, I., Simons, P., and You, J.-H. (2000). Partiality and Disjunctions in Stable Model Semantics. In Cohn, A. G., Giunchiglia, F., and Selman, B., editors, *Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR 2000), April 12-15, Breckenridge, Colorado, USA*, pages 411–419. Morgan Kaufmann Publishers, Inc.

[Keller et al., 2005] Keller, U., Lara, R., Lausen, H., Polleres, A., Predoiu, L., and Toma, I. (2005). Wsmx discovery. Technical report, WSMX. WSMX Working Draft D10v0.2. Available from http://www.wsmo.org/TR/d10/v0.2/.

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

[Lloyd, 1987] Lloyd, J. W. (1987). *Foundations of Logic Programming (2nd edition)*. Springer-Verlag.

[McCarthy, 1986] McCarthy, J. (1986). Applications of circumscription to formalizing common sense knowledge. *Artificial Intelligence*, 28:89–116. Reprinted in [McCarthy, 1990].

[McCarthy, 1990] McCarthy, J. (1990). *Formalization of common sense, papers by John McCarthy edited by V. Lifschitz*. Ablex.

[Motik and Sattler, 2005] Motik, B. and Sattler, U. (2005). Practical DL reasoning over large ABoxes with KAON2. *to be published*.

[Niemelä, 1999] Niemelä, I. (1999). Logic Programming with Stable Model Semantics as Constraint Programming Paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3–4):241–273.

[Pan and Horrocks, 2004] Pan, J. Z. and Horrocks, I. (2004). OWL-E: Extending OWL with expressive datatype expressions. IMG Technical Report IMG/2004/KR-SW-01/v1.0, Victoria University of Manchester. Available from http://dl-web.man.ac.uk/Doc/IMGTR-OWL-E.pdf.

[Pease et al., 2000] Pease, A., Chaudhri, V., Lehman, F., and Farquhar, A. (2000). Practical knowlege representation and the darpa high performance knowledge base project. In *Seventh International Conference on Principles of Knowledge Representation and Reasoning*, Breckenridge, CO.

[Riazanov and Voronkov, 2002] Riazanov, A. and Voronkov, A. (2002). The Design and Implementation of Vampire. *AI Communications*, 15(2-3):91–110.

[Roman et al., 2004] Roman, D., Lausen, H., and Keller, U. (2004). Web service modeling ontology standard (WSMO-standard). Working Draft D2v1.0, WSMO. Available from http://www.wsmo.org/2004/d2/v1.0/.

[Simons et al., 2002] Simons, P., Niemelä, I., and Soininen, T. (2002). Extending and Implementing the Stable Model Semantics. *Artificial Intelligence*, 138:181–234.

[Tsarkov and Horrocks, 2003a] Tsarkov, D. and Horrocks, I. (2003a). DL reasoner vs. first-order prover. In *Proc. of the 2003 Description Logic Workshop (DL 2003)*, volume 81 of *CEUR (`http://ceur-ws.org/`)*, pages 152–159.

[Tsarkov and Horrocks, 2003b] Tsarkov, D. and Horrocks, I. (2003b). Reasoner prototype. implementing new reasoner with datatypes support. Technical report, WonderWeb project.

[Tsarkov et al., 2004] Tsarkov, D., Riazanov, A., Bechhofer, S., and Horrocks, I. (2004). Using vampire to reason with owl. In *Proc. of the 2004 International Semantic Web Conference (ISWC 2004)*. To appear.

[Ullman, 1988] Ullman, J. D. (1988). *Principles of Database and Knowledge-Base Systems, Volume I.* Computer Science Press.

[Ullman, 1989] Ullman, J. D. (1989). *Principles of Database and Knowledge-Base Systems, Volume II.* Computer Science Press.

[Waldinger, 2004] Waldinger, R. (2004). Formulation and validation of the axiomatic semantics of owl. available at: `http://www.ai.sri.com/daml/owl/axiomatic.htm`.

[Wielemaker, 2003] Wielemaker, J. (2003). An overview of the SWI-Prolog programming environment. In Mesnard, F. and Serebenik, A., editors, *Proceedings of the 13th International Workshop on Logic Programming Environments*, pages 1–16, Heverlee, Belgium. Katholieke Universiteit Leuven. CW 371.

[Yang et al., 2003] Yang, G., Kifer, M., and Zhao, C. (2003). FLORA-2: A rule-based knowledge representation and inference infrastructure for the semantic web. In *Proceedings of the Second International Conference on Ontologies, Databases and Applications of Semantics (ODBASE)*, Catania, Sicily, Italy.