IST-2004-027655
ESTRELLA
European project for Standardised Transparent Representations in order to Extend Legal Accessibility

Specific Targeted Research Project
Information Society Technologies

Deliverable 1.5
Reference inference engine for LKIF

Due date of deliverable: 31 December 2006
Actual submission date: 29 January 2007

Start date of project: 1 January 2006  Duration: 30 months

Corvinus University Budapest

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)

| Dissemination Level | | |
|--------------------|-------------|
| PU                 | Public | X |
| PP                 | Restricted to other programme participants (including the Commission Services) |
| RE                 | Restricted to a group specified by the consortium (including the Commission Services) |
| CO                 | Confidential, only for members of the consortium (including the Commission Services) |
Executive Summary

This deliverable reports on the analyses and description of inference mechanisms and tools that are capable of processing knowledge bases using features of the LKIF. Therefore the document is based on the specification of LKIF published in [Boer et al., 2007]. The work presented in this document was carried out under T1.6. The statements and results of this deliverable will be further tested, applied and possibly revised in WP2 test applications. If changes have effect on the core technologies suggested here than the necessary modifications will also have impact on WP4 LKIF refinement.

In this document instead of providing one inference engine to serve the needs of LKIF, we will provide solutions for the different levels of knowledge representations LKIF is based on, in the logical level. These levels are associated with demands of different applications, as they have a growing level of expressive power. We provide examples, describe an inference mechanism applicable, and show reference implementations if available.
Reference inference engine for LKIF
Deliverable 1.5

András Fürhécz and Réka Vas
Corvinus University Budapest

Marcello Di Bello
University of Amsterdam
## Contents

1 Introduction ................................................................. 1  
   1.1 Logical representations ........................................ 1  

2 DLP ................................................................................. 3  
   2.1 Datalog ($L_0$) ........................................................ 4  
   2.2 Datalog with equality ($L_1$) ..................................... 5  
   2.3 Datalog with equality and integrity constraints ($L_2$) .... 6  
   2.4 Prolog ($L_3$) .......................................................... 6  
   2.5 Example for the legal domain ...................................... 7  
   2.6 Complexity and inference algorithm ............................ 8  

3 OWL DL ............................................................................... 10  
   3.1 Expressiveness and Decidability ................................. 11  
   3.2 Example ................................................................. 12  
   3.3 Inference algorithm and implementations .................... 12  

4 $\mathcal{A}\mathcal{L}$-log ............................................................... 14  
   4.1 Syntax and Semantics ............................................... 14  
   4.2 Example ................................................................. 15  
   4.3 Implementation ........................................................ 16  

5 DL-safe rules ...................................................................... 17  
   5.1 Syntax and Semantics ............................................... 17  
   5.2 Expressiveness and Decidability ................................. 18
Chapter 1

Introduction

This deliverable reports on the analyses and description of inference mechanisms and tools that are capable of processing knowledge bases using features of the LKIF. Therefore the document is based on the specification of LKIF published in D1.1. The work presented in this document was carried out under T1.6. The statements and results of this deliverable will be further tested, applied and possibly revised in WP2 test applications. If changes have effect on the core technologies suggested here than the necessary modifications will also have impact on WP4 LKIF refinement.

Legal reasoning is based on argumentation, however the first step is always building a knowledge base, representing the information needed for argumentation schemes. If we would like to model legal material in its full detail in arguments, we have to use higher-order modal logics, but then tractable reasoning is not attainable. The working hypothesis of Estrella is that description logic is often sufficient for modelling information of the legal domain.

Description logic semantically being a subset of first-order logic, does not support plausible inference, but will serve inference services for representing the logical layer of argumentation. As the first step of legal reasoning this kind of inference would extend the available set of knowledge, as a basis for building argumentation.

In this document instead of providing one inference engine to serve the needs of LKIF, we will provide solutions for the different levels of knowledge representations LKIF is based on, in the logical level. These levels are associated with demands of different applications, as they have a growing level of expressive power. We will provide examples, describe an inference mechanism applicable, and show reference implementations if available.

1.1 Logical representations

OWL DL has been chosen as a central knowledge representation for LKIF. Some of the most important factors doing so are advantages from tractable inference services:
decidable and tractable;
Decidability is very important for many applications. We will discuss complexity, and empirical running times of reasoning engines.

description logic is convenient to model Real World;
There is a lot of experience in ontology development already available, DL is more comprehensible than rule-based systems.

there are lots of available tools for inference.

We would detail proposed logical representations with growing expressive power, regarding inference services. Every subsection has the same structure:

- expressive power of formalism, what can be described
- inference mechanism applicable
- example for application in the legal domain
- reasoning engine implementation (if available)

We will provide examples taken from the Act for Value Added Tax. If possible, the examples will be some specialized, more expressive form of the same material showed previously.

The representations are the following restricted or extended forms of OWL DL:

**DLP** is a restricted form of OWL DL which is also compatible with rule systems;

**OWL DL** is the most popular description logic $\mathcal{SHOIN}(D)$, developed for the Semantic Web;

**$\mathcal{AL}$-log** is a limited rule extension for OWL DL;

**DL-safe rules** being the most expressive but still decidable subset of SWRL.

$\mathcal{AL}$-log has been included because there are no available tractable reasoning engines for DL-safe rules at the moment. We expect new tools to be developed in the near future, but in the meantime $\mathcal{AL}$-log can be an alternative building and evaluating legal knowledge bases in practice.
Chapter 2

DLP

OWL DLP is one name for the DLP subset of OWL. Although it is not one of the sublanguages formally defined in the OWL W3C standard (OWL Lite, OWL DL, and OWL Full), users of OWL need only to follow a few simple guidelines to assure their ontologies are within the DLP subset.

Description Logic Programs (DLPs) are ontological knowledge bases which lie within the intersection of OWL and Logic Programming. They could be described as a naive Horn fragment of OWL DL. They are also a fragment of the more expressive Horn-SHIQ, which is more powerful but follows similar intuitions [Vrandecic et al., 2005].

DLP imposes certain constraints on OWL DL in order to guarantee that all axioms stated are transformable in an efficient way to Horn clauses, i.e. rules in the sense of traditional logic programming, but it does not define new semantics or syntax. Every OWL DLP ontology is an OWL DL ontology as well. In other words DLP is a proper subset of OWL DL, that is able to express the following features of OWL-DL:

1. concept disjointness,
2. domains and ranges of properties,
3. inverse and symmetric properties,
4. functional and inverse-functional properties,
5. subproperty and equivalence relations between object properties,
6. transitive properties, and
7. a limited form of General Concept Inclusion axioms (GCIs).

The constraints imposed on DLP regarding the more expressive OWL DL, lead to DLP enjoying a much better data complexity (polynomial) and combined complexity (EXPTime). This allows expecting more efficient and responsive tools than full OWL DL reasoning will ever be able to achieve due to the complexity of DL reasoning algorithms.
OWL DLP is not easily defined, but it is possible to give an easy to use sublanguage of OWL DL by listing all the constructors one may use freely in an OWL ontology without the risk of leaving OWL DLP. The allowed OWL DL constructors are the following according to [Vrandecic et al., 2005]: Class, Thing, subClassOf, Property, subPropertyOf, domain, range, Individual, equivalentClass, equivalentProperty, sameAs, differentFrom, AllDifferent, ObjectProperty, DatatypeProperty, inverseOf, TransitiveProperty, SymmetricProperty, FunctionalProperty, InverseFunctionalProperty, intersectionOf.

However this is by no means a description of the whole DLP language. Description Logic Programs are a family of description logics $\mathcal{L}_i$ with different expressiveness and complexity. We will briefly introduce these languages based on [Volz, 2004a] and give an example for the usage of DLP in the legal domain.

2.1 Datalog ($\mathcal{L}_0$)

Datalog denoted by $\mathcal{L}_0$ represents the simplest DLP language, which allows the construction of class equivalence axioms that can be translated into Datalog programs. Two extensions allow certain class descriptions that can only appear on the left hand side ($\mathcal{L}_0^L$) or the right hand side ($\mathcal{L}_0^R$) of DL class inclusion axioms.

Class descriptions in $\mathcal{L}_0$ have the following syntax:

- atomic class — $A$
- conjunction — $C \sqcap D$
- hasValue — $\exists R.\{o\}$

$\mathcal{L}_0^R$ class descriptions are formed according to the syntax of $\mathcal{L}_0$ and the additional description:

- value restriction — $\forall R.C$

$\mathcal{L}_0^L$ class descriptions are formed according to the syntax of $\mathcal{L}_0$ and the additional descriptions:

- disjunction — $C \sqcup D$
- existential restriction — $\exists R.C$

Definition 1 (DLP knowledge base). A DLP TBox $\mathcal{T}_0^{DLP}$ is a set of DLP TBox axioms
Section 2.2  Datalog with equality ($\mathcal{L}_1$)

Extending Datalog with class equivalence axioms leads to the DLP language $\mathcal{L}_1$. Class descriptions in $\mathcal{L}_1$ extend the set of descriptions in $\mathcal{L}_0$ by the following:

\[
\begin{align*}
\text{one-of with arity 1} & \quad \{o\} \\
\text{Universal class} & \quad \top \\
\text{min. cardinality 0} & \quad \geq 0 \, R
\end{align*}
\]

where $C$ is a $\mathcal{L}_0^C$ class, $D$ is a $\mathcal{L}_0^R$ class, $E$ and $F$ are $\mathcal{L}_0$ classes and $P, Q$ are properties.

A DLP ABox $A_{\mathcal{DLP}}^0$ is a set of axioms of the form:

\[
\begin{align*}
\text{property fillers} & \quad P(a, b) \\
\text{individual assertion} & \quad D(a)
\end{align*}
\]

where $P$ is a property, $D$ is a $\mathcal{L}_0^R$ class and $a, b$ are individuals.

A DLP knowledge base $KB_{\mathcal{DLP}}^0$ is a pair $\langle T_{\mathcal{DLP}}^0, A_{\mathcal{DLP}}^0 \rangle$.

There are several alternatives available to assign semantics to the class descriptions of the DLP language $\mathcal{L}_0$ and the following $\mathcal{L}_i$ languages. The easiest way is to provide a reduction to some other logic. Since we will define syntactic subsets of $\mathcal{SHOIN}(D)$, DLP can inherit the semantics of OWL DL as obvious.

A Datalog ($\mathcal{L}_0$) KB can include class and property hierarchy, class conjunction and hasValue restrictions. Additionally, the DLP TBox associated with $\mathcal{L}_0$ allows to express most of the OWL DL property attributes such as property inverse, symmetric and transitive properties. $\mathcal{L}_0^R$ is able to express value restrictions in necessary conditions for class membership, similarly $\mathcal{L}_0^R$ allows existential restrictions to sufficient class membership conditions.

\[
\begin{align*}
\text{class} & \quad \{ \text{inclusion} \quad C \subseteq D \\
\text{equivalence} & \quad E \equiv F \\
\text{subsumption} & \quad P \subseteq Q \\
\text{equivalence} & \quad P \equiv Q \\
\text{inverse} & \quad P \equiv Q^- \\
\text{symmetry} & \quad P \equiv P^- \\
\text{transitivity} & \quad P^+ \subseteq P \\
\text{domain} & \quad \top \subseteq \forall P^- . D \\
\text{range} & \quad \top \subseteq \forall P . D
\end{align*}
\]

A Datalog ($\mathcal{L}_0$) KB can include class and property hierarchy, class conjunction and hasValue restrictions. Additionally, the DLP TBox associated with $\mathcal{L}_0$ allows to express most of the OWL DL property attributes such as property inverse, symmetric and transitive properties. $\mathcal{L}_0^R$ is able to express value restrictions in necessary conditions for class membership, similarly $\mathcal{L}_0^R$ allows existential restrictions to sufficient class membership conditions.

\[
\begin{align*}
\text{class} & \quad \{ \text{inclusion} \quad C \subseteq D \\
\text{equivalence} & \quad E \equiv F \\
\text{subsumption} & \quad P \subseteq Q \\
\text{equivalence} & \quad P \equiv Q \\
\text{inverse} & \quad P \equiv Q^- \\
\text{symmetry} & \quad P \equiv P^- \\
\text{transitivity} & \quad P^+ \subseteq P \\
\text{domain} & \quad \top \subseteq \forall P^- . D \\
\text{range} & \quad \top \subseteq \forall P . D
\end{align*}
\]

where $C$ is a $\mathcal{L}_0^C$ class, $D$ is a $\mathcal{L}_0^R$ class, $E$ and $F$ are $\mathcal{L}_0$ classes and $P, Q$ are properties.
Class expressions on the right hand side of class inclusion axioms ($L^R_1$) are extended with:

\[
\text{max. cardinality } 1 \quad \leq 1 \quad R
\]

Class expressions on the left hand side of class inclusion axioms ($L^L_1$) are extended with:

\[
\text{min. cardinality } 1 \quad \geq 1 \quad R
\]
\[
\text{one-of with arity } n \quad \{o_1, \ldots, o_n\}
\]

Individual equivalence is allowed in the ABox, introducing the axiom $a = b$ where $a$ and $b$ are individuals.

The main contribution in expressiveness compared to $L_0$ is the singleton set constructor and the equality of individuals, which introduces nominals in the terminology of the TBox. The maximum cardinality restriction with value one defines functional properties. They capture the uniqueness aspect of primary keys in relational databases.

### 2.3 Datalog with equality and integrity constraints ($L_2$)

$L_2$ introduces atomic class negation and individual inequality compared to $L_1$. Class expressions on the right hand side of class inclusion axioms ($L^R_2$) are extended with:

\[
\text{atomic negation} \quad \neg A
\]
\[
\text{empty class} \quad \bot
\]
\[
\text{max. cardinality } 0 \quad \leq 0 \quad R
\]
\[
\text{property has filler that belongs to } \bot \quad \exists R. \bot
\]

Integrity constraints allow to define basic constraints on classes, and allow classes to become unsatisfiable for the first time through inclusion axioms. $L_2$ is able to catch OWL DisjointClass axiom, since $\bot$ can appear on the right side of an inclusion axiom.

### 2.4 Prolog ($L_3$)

There is a forth level of Description Logic Programs called Prolog ($L_3$) which allows the construction of class equivalence axioms that can be translated into Prolog programs. $L_3$ greatly increases the expressive power by allowing existential property restrictions on the right-hand side of inclusion axioms.
However Prolog itself is undecidable, since it can express all recursively enumerable predicates. Hence the complexity of $\mathcal{L}_3$ is not given by its translation into Prolog but rather by a translation into well-known Description Logics. Since $\mathcal{L}_3$ allows (restricted) use of language features in $\mathcal{SHOIN}$, we can conjecture that $\mathcal{L}_3$ has at most the same complexity as $\mathcal{SHOIN}$, i.e. $\text{NExpTime}$.

If we translate a $\mathcal{L}_3$ DLP program to Prolog, obviously we do not meet our goal of tractability anymore. Since inference on OWL DL is expected to be more efficient, the $\mathcal{L}_3$ subset of OWL DL is not beneficial when expressiveness and complexity are of high concern.

## 2.5 Example for the legal domain

Now we will provide a demonstration of inference in Description Logic Programs. We will show a part of a knowledge base describing the Act for Hungarian Value Added Tax, and have a look at the inference services provided by DLP.

A Datalog ($\mathcal{L}_0$) KB can include class and property hierarchy, class conjunction, $\text{hasValue}$ restrictions and property attributes. Now we try to model the Material scope of VAT described in Section 3 of Hungarian Act for Value Added Tax using $\mathcal{L}_0$:

\[
\text{Section 3.}
\]

Based on the provisions of this Act, the following shall be subject to value added tax (hereinafter referred to as ‘tax’):

a) goods and services supplied for consideration within the territory of the country by a taxable person, and

b) goods purchased within the European Communities (hereinafter referred to as ’Communities’), and

c) the importation of goods.

The main concepts of the legal domain introduced in our example knowledge base are $\text{Place}$, $\text{Activity}$, $\text{Actor}$ and $\text{Goods}$. Section 3/b of VAT can be described in $\mathcal{L}_0$ using the following terms:

\[
\begin{align*}
\text{EconomicActivity} & \sqsubseteq \text{Activity} \\
\text{SupplyOfGoods} & \sqsubseteq \text{EconomicActivity} \\
\text{SupplyOfGoods} \sqcap & \exists \text{placeOfAcquisition}.\text{Place} \sqsubseteq \text{AcquisitionOfGoods} \\
\text{AcquisitionOfGoods} \sqcap & \exists \text{placeOfAcquisition}.\text{CommunityTerritory} \sqsubseteq \text{CommunityAcquisition} \\
\text{CommunityAcquisition} & \sqsubseteq \text{MaterialScope}
\end{align*}
\]
Now individuals of *Activity* falling into the case described in Section 3/b of VAT should be categorized into *CommunityAcquisition*. E.g. if we have the following assertions in the ABox (introducing individuals *Budapest* and *buyFlatInBudapest*):

\[
\text{CommunityTerritory}(\text{Budapest}) \\
\text{SupplyOfGoods}(\text{buyFlatInBudapest}) \\
\text{placeOfAcquisition}(\text{buyFlatInBudapest}, \text{Budapest})
\]

A reasoner can infer *MaterialScope*(buyFlatInBudapest), which means buying a flat in Budapest is an activity covered by Value Added Tax, as expected.

However if we would like to infer activities not covered by the same concept, we have to extend the knowledge base. For example here is an acquisition not covered by Section 3/b of VAT:

\[
\text{Canada} \sqsubseteq \neg \text{CommunityTerritory} \\
\text{Canada}(\text{Toronto}) \\
\text{SupplyOfGoods}(\text{buyCornInToronto}) \\
\text{placeOfAcquisition}(\text{buyCornInToronto}, \text{Toronto})
\]

We would like to conclude \(\neg \text{CommunityAcquisition}(\text{buyCornInToronto})\). The main deficiency in expressiveness here is that we can’t use class equivalence when *CommunityAcquisition* is defined, as existential restriction is not allowed inside class equivalence axioms. Some other axioms can be expressed in DLP according to this example. Specifying *placeOfAcquisition* as functional is possible in \(L_1\), desirable because of the Open World Assumption (if an action has one place of acquisition in Toronto does not lead to the conclusion all places are outside the Community):

\[
\text{AcquisitionOfGoods} \sqsubseteq \leq 1 \text{placeOfAcquisition}
\]

The DisjointClass axiom introduced in \(L_2\) can help to describe a necessary condition for *CommunityAcquisition*, but this is far from intuitive:

\[
\text{OutsideCommunity} \cap \text{CommunityTerritory} \sqsubseteq \bot \\
\text{CommunityAcquisition} \cap \exists \text{placeOfAcquisition.OutsideCommunity} \sqsubseteq \bot
\]

### 2.6 Complexity and inference algorithm

Reasoning on DLP is based on the translation of DL axioms into Logic Programming (LP) rules [Volz, 2004a]. Typical DL TBox and ABox queries like class subsumption, class satisfiability, instance checking and instance retrieval are all reduced to
Section 2.6 Complexity and inference algorithm

operations on the logic program. In particular, DL TBox reasoning tasks are expensive to handle, since their reduction involves updates on the logic program. On the other hand, DL ABox reasoning tasks can be easily reduced to (multiple) LP queries.

The Datalog ($L_0$) and Datalog with negation ($L_1$) DLP languages can be implemented in Datalog variants, yielding to a polynomial of the fourth degree as an upper bound for the complexity of ABox reasoning tasks. The complexity of TBox reasoning problems has an ExpTime upper bound.

There is no widely used OWL DLP reasoning engine available. An implementation for converting DLP ontologies into logic programs is available as dlpconvert [Motik et al., 2005b]. dlpconvert$^1$ is based on the algorithms for reducing description logics to Datalog implemented in KAON2. It reads in an OWL ontology, reduces it to disjunctive Datalog and finally serializes it into a logic program, which can be used as input for Prolog interpreters or for easier reading and thus understanding by people with an appropriate logic background.

$^1$dlpconvert, http://logic.aifb.uni-karlsruhe.de/dlpconvert/
Chapter 3

OWL DL

OWL in general is syntactically layered on RDF, extending it with additional vocabulary, but it differs from Description Logics in several ways:

1. N-Ary language constructs such as conjunction ($\cap$) have to be encoded into several RDF statements;
2. OWL graphs can contain circular syntactic structures which are not possible in Description Logics;
3. due to the circular meta-model of RDF, classes and properties can be made instances of themselves. For example, it would be possible to assert the statement $\text{rdf:type}$(owl:Class,owl:Class);
4. the identifiers used for classes, properties and individuals do not have to be disjoint.

OWL has three increasingly expressive sublanguages: OWL Lite, OWL DL, and OWL Full. OWL DL supports those users who want the maximum expressiveness while retaining computational completeness (all conclusions are guaranteed to be computable) and decidability (all computations will finish in finite time). OWL DL includes all OWL language constructs, but they can be used only under certain restrictions [McGuinness and van Harmelen, 2004]. So OWL DL is a subset of OWL Full that makes the following restrictions:

1. the OWL vocabulary can not be used as identifiers for classes, properties or individuals;
2. class constructors with syntactic cycles are disallowed;
3. the sets of identifiers used for classes, properties, and individuals must be disjoint;
4. cardinality restrictions may not be stated on properties which are transitive.
3.1 Expressiveness and Decidability

In particular, OWL DL is based on the $\mathcal{SH}$ family of Description Logics [Horrocks et al., 2000], to achieve a suitable balance between computational complexity of inference and expressiveness requirements identified in [Hefflin, 2004]. The constructors and axioms supported by $\mathcal{SH}$ include the traditional Boolean connectives (intersection, union and complement), restrictions on properties, transitive properties and a property hierarchy. The property hierarchy is important for OWL, as it is a feature of RDFS, while transitive properties have been identified as an important requirement in many applications [Hefflin, 2004].

Members of the $\mathcal{SH}$ family include the influential $\mathcal{SHI}Q$ Description Logic [Horrocks et al., 1999], which adds inverse properties and generalized cardinality restrictions, and $\mathcal{SHO}Q(D)$ [Horrocks and Sattler, 2001], which adds the ability to define a class by enumerating its instances (e.g., the class \{Monday, Tuesday, Wednesday, Thursday, Friday\}) and support for datatypes and values.

We will not provide the full semantics of OWL DL here, but it is included in Deliverable 1.1.

The integration of datatypes in OWL is again heavily influenced by Description Logic research, which has pointed out that careless use of datatypes may cause complexity blowups or undecidability [Lutz, 2002].

In the $\mathcal{SHO}Q(D)$ Description Logic a solution was provided for this problem. It was proved that strictly separating the interpretation of datatypes and values from that of classes and individuals can help to avoid complexity and decidability problems. The separation is further strengthened by dividing properties into two disjoint sets of abstract and datatype properties. This design has the advantage that reasoning with datatypes and values can be almost entirely separated from reasoning with classes and individuals. Moreover, the language remains decidable if datatype and value reasoning is decidable.

OWL DL was carefully designed to remain decidable, and does not include, for example, relationships between role chains, which would cause undecidability. At the same time this does not mean that inference in OWL DL is easy. OWL DL has a difficult entailment problem, as inference $\mathcal{SHO}IN(D)$ is of worst-case nondeterministic exponential time ($\text{NExpTime}$) complexity [Tobies, 2001], and OWL DL should have the same complexity. Another major problem is that there is no known complete (and practical) algorithm for inference in $\mathcal{SHO}IN(D)$, i.e., one that is likely to perform well on the kinds of problems encountered in typical applications. In default of such an algorithm, the behavior of OWL DL reasoners is likely to be less predictable (both in terms of the time taken to respond to queries, and the use of system resources), and they may sometimes return the answer ‘Unknown’ in response to queries [Horrocks et al., 2003].
3.2 Example

One of the most important advantages to use OWL DL instead DLP is that necessary and sufficient conditions can be easily described for classes, as the usage of class equivalence is not limited to a restricted set of DL expressions. E.g. in the previous example $\text{CommunityAcquisition}$ can be directly defined as:

$$\text{AcquisitionOfGoods} \sqcap \exists \text{placeOfAcquisition}.\text{CommunityTerritory} \equiv \text{CommunityAcquisition}$$

Another limitation, the lack of disjunction on the right side of class inclusion axioms leads often to troubles. For example a taxable person as described in Section 4(1) in VAT:

‘Taxable person’ shall mean any natural or legal person or unincorporated organization who (which) may, in its own name, obtain rights, assume obligations, sue, and perform economic activities on its own behalf, regardless of the purpose and results thereof.

This means ‘taxable person’ is a subclass of the union of three classes:

$$\text{Taxable} \sqsubseteq \text{NaturalPerson} \sqcup \text{LegalPerson} \sqcup \text{UnincorpOrg}$$

3.3 Inference algorithm and implementations

Since OWL DL suffers from a very high computational complexity, and efficient reasoners able to deal with them in a sufficiently scalable way are still one of the main challenges for the community.

There are basically two different approaches for OWL DL reasoning. The most appropriate implementations are based on the tableau calculi [Horrocks et al., 2000].

For refutation tableaux, the objective is to show that the negation of a formula cannot be satisfied. There are rules for handling each of the usual connectives, and applying these rules the subtableau may be divided into two. If any branch of a tableau leads to an evident contradiction, the branch closes. If all branches close, the proof is complete and the original formula is a logical truth.

More specifically, a tableau calculus consists of a finite collection of rules with each rule specifying how to break down one logical connective into its constituent parts. If there is such a rule for every logical connective then the procedure will eventually
produce a set which consists only of atomic formulae and their negations, which cannot be broken down any further.

The most prominent implementations following the tableau calculi are Pellet\textsuperscript{1}, RacerPro\textsuperscript{2} ([Haarslev and Möller, 2001]) and Fact++\textsuperscript{3}.

Another approach is to reduce the knowledge base to a disjunctive datalog program, as with DLP. However it is not yet clear but likely that this can not be achieved for the whole OWL DL language. KAON2\textsuperscript{4} is an OWL DL reasoning engine following this path, also support for nominals is still missing, and large numbers in cardinality statements can lead to severe decrease in performance.

An in-depth comparison of these reasoning engines can be found in [Gardiner et al., 2006]. When support, open solutions and being widespread are taken into consideration in the first place, Pellet is an absolute winner. Fortunately in respect of performance Pellet is also not behind even commercial solutions like RacerPro.

\footnotesize
\begin{itemize}
\item \textsuperscript{1}Pellet: An OWL DL Reasoner, \url{http://pellet.owldl.com/}
\item \textsuperscript{2}RacerPro, \url{http://www.racer-systems.com/}
\item \textsuperscript{3}Fact++, \url{http://owl.man.ac.uk/factplusplus/}
\item \textsuperscript{4}KAON2, Ontology Management for the Semantic Web, \url{http://kaon2.semanticweb.org/}
\end{itemize}
Chapter 4

\textit{\textbf{\textit{AL}}-log}

\textit{\textbf{AL}}-log is a two-component system, named after Attribute Language and Datalog, whose main feature is to add to the framework of deductive databases the structuring power of description logics.

The original \textit{\textbf{AL}}-log knowledge representation described in [Donini et al., 1998] combines a structural subsystem: the \textit{\textbf{ALC}} description logic, and a relational subsystem: the database language Datalog [Ceri et al., 1989]. The interaction between the two is realized by allowing constraints for Datalog clauses expressed in \textit{\textbf{ALC}}. Constraints on variables specify them to range over the instances of a concept, while constraints on individuals require them to belong to a specified concept.

The hybrid reasoner built for \textit{\textbf{AL}}-log computes answers to queries based on both the structural and relational component of the knowledge base. The set of conclusions the reasoner can draw is not just the union of the conclusions derivable from each representation, because the two subsystems can interact.

4.1 Syntax and Semantics

At first we introduce the Datalog language and give a brief description of the semantics [Parsia et al., 2005]. The syntax of Datalog is as follows:

\textbf{Definition 2} (Datalog Syntax). Let $V_x$ be a set of variables, $V_{cons}$ a set of constants, and $V_{rp}$ a set of predicate symbols. A \textit{Datalog term} is any variable from $V_x$ or constant symbol from $V_{cons}$. A \textit{Datalog atom} is of the form $q(t_1,\ldots,t_n)$, where $q$ is a predicate symbol of arity $n \geq 0$ in $V_{rp}$ and $t_1,\ldots,t_n$ are datalog terms. A rule $r$ is of the form:

\[ a \leftarrow b_1, \ldots, b_k \]

Where $a, b_1, \ldots, b_k$ are datalog atoms. The atom $a$ is the head (consequent) of the rule, while $b_1, \ldots, b_k$ is the body (antecedent) of $r$. A Datalog program $P$ is a finite set of rules.
A model-theoretic semantics for Datalog can be given by means of interpretations, which assigns meaning to constant symbols and predicated. The interpretation $I$ satisfies the antecedent $A(r)$ if $A$ is empty or $I$ satisfies every Datalog atom in $A(r)$. The interpretation $I$ satisfies the consequent $H(r)$ if it is non-empty or $I$ satisfies every Datalog atom in $H(r)$. The interpretation $I$ satisfies the Datalog rule $r$ iff whenever $I$ satisfies the antecedent, it also satisfies the consequent [Parsia et al., 2005].

Now as we have defined Datalog, we can introduce a general formalism that combines the description logic $SHOIN(D)$ with the Datalog relational subsystem.

The general idea is to let Datalog atoms be built of object or datatype terms of the description logic. Also DL atoms are defined as usual from classes, object properties, datatype properties and datatypes. Rules can have DL atoms and Datalog atoms both in the antecedent and in the consequent. A combined knowledge base consists of a $SHOIN(D)$ knowledge base and a finite set of rules. The exact semantics for such system is described in [Parsia et al., 2005].

Such a combined knowledge representation is not decidable however, as e.g. it supersedes SWRL. The following rule extension proposals for OWL DL are all a decidable subset of the general combined knowledge representation, and have different expressiveness and computational properties.

A slightly generalized form of the original $ALC$-log language will be provided here. First, the description logic used as the basis for the representation is $SHOIN(D)$ instead of $ALC$, introducing all the features of OWL DL. Secondly the usage of OWL datatypes and SWRL built-ins is also allowed in the antecedent of Datalog rules.

**Definition 3.** An $ALC$-log knowledge base is a combined knowledge base $K = (\Sigma, P)$, where $\Sigma$ is a $SHOIN(D)$ knowledge base and $P$ is a set of Datalog rules. DL atoms in $P$ appear only in the antecedent of the rules, and are only of the form $C(t)$, $D(v)$, or $builtin(q, v_1, \ldots, v_n)$, where $C$ is a $SHOIN(D)$ class, $D$ an OWL datatype, $t$ is an object term, $q$ is a built-in procedure and $v, v_1, \ldots, v_n$ are datatype terms.

### 4.2 Example

The primary advantage of rules is we can transfer values through properties. Suppose we have the concepts LegalPerson and Contract in our ontology. The following rule describes contractualRelation as the relation between contracting parties:

\[
contractualRelation(?x, ?y) : -
\]

\[
Contract(?c), obligor(?c, ?x), obligee(?c, ?y),
\]

\[
LegalPerson(?x), LegalPerson(?y).
\]
However this rule is in $\mathcal{AL}$-log when the only DL atoms are class atoms $\text{Contract}$ and $\text{LegalPerson}$. Included relations $\text{contractualRelation}$, $\text{obligor}$ and $\text{obligee}$ are datalog predicates, not DL properties. We can not state e.g. $\text{obligor}$ and $\text{obligee}$ as mandatory for every $\text{Contract}$:

$$\text{Contract} \sqsubseteq \geq 1 \text{obligor} \sqcap \geq 1 \text{obligee}$$

### 4.3 Implementation

The inference algorithm for hybrid deduction in $\mathcal{AL}$-log is based on constrained SLD-derivation and constrained SLD-refutation, as described in the original paper [Donini et al., 1998], but only for the $\mathcal{ALC}$ description logic. A prototype implementation was created by Parsia et. al ([Parsia et al., 2005]) coupling the Pellet OWL DL reasoner with Prolog, thus extending the algorithm to full $\mathcal{SHOIN}$(D).

They have also created a lightweight knowledge base editor Swoop+Rules\(^1\) not only providing reasoning capabilities but also simple ontology management tools for OWL DL ontologies with rules extension. The tool can also classify rules into categories of logical representation: DL syntactic sugar, $\mathcal{AL}$-log, CARIN, DL-safe and SWRL subsets are identified.

Chapter 5

DL-safe rules

In this chapter, OWL DL is extended with DL-safe rules. Note that the arbitrary extension of OWL DL with rule leads to undecidability, hence the need of safety conditions [Motik et al., 2005a]. The combination of OWL DL and DL-safe rules gives a decidable subset of SWRL (Semantic Web Rule Language).

5.1 Syntax and Semantics

We first define the syntax of the rule part of SWRL under safety conditions:

Definition 4 (DL-safe rules Syntax). Let KB be an OWL DL knowledge base. Let \(V_{\text{cons}}\) a set of constants and \(V_{\text{rp}}\) a set of predicates that occur in KB. A DL-atom is of the form \(A(s)\) or \(R(s, s')\) where \(A\) in \(V_{\text{rp}}\) is a one place predicate and \(R\) in \(V_{\text{rp}}\) a two place predicate, and \(s, s'\) are constants. We have:

- A rule \(r\) is called DL-safe iff each variable in \(r\) occurs in a non DL-atom in the antecedent of the rule.
- A set of rules (i.e., a program \(P\)) is DL-safe iff all its rules are DL-safe.
- A safe-DL knowledge base is a tuple \(\langle KB, P \rangle\) where \(KB\) is a OWL DL knowledge base and \(P\) is a set of DL-safe rules.
- The combination of OWL DL with DL-safe rules is called the safe subset of SWRL, or simply safe SWRL.

The SWRL vocabulary is the OWL DL vocabulary \(V\) with the addition of the sets:

- \(V_{\text{IX}}\) for individual variables, denoted by \(x, y, z\);
- \(V_{\text{DX}}\) for data type variables, denoted by \(m, n\);
- \(V_{\text{built-in}}\) for built-in names.
It is useful to introduce two notational shortcuts. Any variable in $V_X$ or name in $V_I$ will be called object term, and denoted by $t$, with indices if necessary. Likewise, a datatype term, denoted by $v$, is either a literal in $V_L$ or a datatype variable in $V_{DX}$.

The SWRL logical vocabulary extends the OWL logical vocabulary with $\rightarrow$ and $\land$. Strictly speaking, since SWRL can be translated into FOL, its logical vocabulary is equivalent to that of FOL, namely truth-functional connectives ($\land, \lor, \rightarrow, \neg$) and quantifiers ($\exists, \forall$).

**Definition 5.** The set of SWRL atoms is defined by the following rule:

$$SWRL\text{-}Atom ::= C(t) | D(v) | P(t_1,t_2) | T(t,v) | t_1 = t_2 | t_1 \neq t_2$$

where $C$ is a OWL class (atomic or complex); $P \in V_{IP}$ a OWL atomic property, $D \in V_{DC}$ a OWL datatype class; and $T \in V_{DP}$ a OWL datatype property.

**Definition 6.** The set of SWRL rules is the smallest set constructed out of the SWRL atoms such every element has the shape:

$$A_1 \land \ldots \land A_2 \rightarrow A,$$

where $A_i$’s and $A$ are SWRL atoms. $A$ is the head (consequent) of the rule; and the (possibly empty) finite conjunction $A_1 \land \ldots \land A_k$ is the body (antecedent) of the rule. Rules will be denoted by $r$. There are $k$ universal quantifiers that scope over the entire rule and that binds the variables in the rule. They can be ordered as one pleases, since they are only universal quantifiers, and there is no harm is swapping them.

**Definition 7.** The set of safe SWRL formulas is the smallest that includes the sets:

- OWL classes (i.e., their FOL translation);
- OWL axioms and facts (i.e., their FOL translation);
- safe SWRL rules or DL-safe rules.

Finally, the semantics of safe SWRL is standard First-Order Logic semantics. In fact, any OWL DL knowledge can be easily translated into FOL (cf. [Volz, 2004b]), while the DL-safe rules are already written in FOL.

### 5.2 Expressiveness and Decidability

The main inference we are interested in safe is query answering, namely deciding whether $KP \cup P \models A$, where $A$ is a ground atom. In [Motik et al., 2005a], Theorem 1, it is shown that query answering in safe SWRL is decidable. Note, however that this decidability results is restricted to OWL DL without concrete domain (e.g. no data-types can be used). Fortunately, the decidability result is extended to OWL DL with concrete domain by Rosati in [Rosati, 2006].
However, decidability is obtained by means of restricting the expressive power. To see how much expressiveness we loose by adopting safe SWRL, let us look at an extended example. Consider a knowledge base $\mathcal{KB}$, where $\mathcal{A}$, $\mathcal{T}$, $\mathcal{P}$ as usual denote assertive and terminological knowledge base, and a program. We mix OWL DL syntax for axioms and facts with FOL syntax for rules. Assume we have:

$$C(a) \in \mathcal{A}$$

$$[C \sqsubseteq D] \in \mathcal{T}$$

$$[\forall x, y : C(x) \land D(y) \rightarrow R(x, y)] \in \mathcal{P}$$

The above rule is not safe since the variables $x, y$ do not occur in a non OWL atom in the body of the rule. That is the kind of limitation that safe SWRL will confront us with. At first, this is a very annoying limitation. However, the first discouragement can be wiped out if some syntactic tricks are used, as presented in [Motik et al., 2005a].

The trick consists of two things: (a) adding a ad hoc non-OWL atom, call it $O$, to the antecedent of the rule, and (b) adding $O(a)$ to $\mathcal{A}$, for every individual name $a$. Thus, we have, call it $\mathcal{KB}'$:

$$O(a), C(a) \in \mathcal{A}$$

$$[C \sqsubseteq D] \in \mathcal{T}$$

$$[C(x) \land D(y) \land O(x) \land O(y) \rightarrow R(x, y)] \in \mathcal{P}$$

Now the strongly safety condition is met. What is the semantic difference that the addition of the $O$-atom gives rise to? Does the trick affect the inferences we can make? The answer is ‘yes and no’. It does not, if we are making inferences about named individuals (individual which we refer to by individual names). It does, if we are making inferences about unknown (unnamed) individuals.

### 5.3 Example

Taking the same example as in section 4.2, but now with DL-safe rules:
\[ contractualRelation(x, y) : \neg Contract(c), \text{obligor}(c, x), \text{obligee}(c, y), \]
\[ \text{LegalPerson}(x), \text{LegalPerson}(y), \]
\[ O(c), O(x), O(y). \]

Now all entities in the rule are DL atoms. We can state e.g. \text{obligor} and \text{obligee} as mandatory for every \text{Contract}, as these are properties and concepts in the TBox:

\[ \text{Contract} \sqsubseteq \geq 1 \text{obligor} \sqcap \geq 1 \text{obligee} \]

Also the consequent of the rule contains DL atoms, thus

\[ contractualRelation \sqsubseteq relation \]

is a proper DL axiom.

### 5.4 Implementation

A proper implementation for DL-safe SWRL is not yet available. The current release of Pellet features experimental DL-safe rules support extending the tableaux reasoner as described in [Kolovski et al., 2006]. However due to performance issues this implementation can only be adopted in research environments. As DL-safe rules are in the focus of investigation recently, we expect tractable solutions emerging soon.
Bibliography


