Rule Language

WonderWeb: Ontology Infrastructure for the Semantic Web

Ian Horrocks[1], Raphael Volz[2,3], Stefan Decker[4], Benjamin Grosof[5], Boris Motik [3]

[1]University of Manchester
Information Management Group
Manchester, Great Britain
email: horrocks@cs.man.ac.uk

[2]University of Karlsruhe
Institute AIFB
D-76128 Karlsruhe
email: volz@aifb.uni-karlsruhe.de

[3]FZI - Research Center for Information Technologies
Haid-und-Neu-Strasse 10-14
D-76131 Karlsruhe
email: {lastname}@fzi.de

[4] University of Southern California
Information Sciences Institute (ISI)
Cambridge, MA, USA
email: stefan@isi.edu

[5]Massachusetts Institute of Technology (MIT)
Sloan School of Management
Cambridge, MA, USA
email: bgrosof@mit.edu

Identifier	Del 2
Class	Deliverable
Version	1.0
Date	31-6-03
Status	Final
Distribution	Internal
Lead Partner	VUM

WonderWeb Project

This document forms part of a research project funded by the IST Programme of the Commission of the European Communities as project number IST-2001-33052.

For further information about WonderWeb, please contact the project co-ordinator:

Ian Horrocks
The Victoria University of Manchester
Department of Computer Science
Kilburn Building
Oxford Road
Manchester M13 9PL
Tel: +44 161 275 6154
Fax: +44 161 275 6236
Email: wonderweb-info@lists.man.ac.uk

Contents

Executive Summary

This deliverable describes the progress on the development of a rule layer in the WonderWeb project. For the expression of rules, we build on Datalog, which represents a well-studied language with known formal properties. Unlike other approaches we seek a tight integration of the ontology language and the rules language. This is achieved via a translation of ontologies described in Description Logics into a set of rules described in the Datalog language. We present a first implementation and evaluation that shows the suitability of our approach.

1 Introduction

Description Logic has become one of the most prominent knowledge representation formalisms, and with recent work on the Ontology Web Language (OWL) standard [22], it is likely to be the basic KR in the Semantic Web context.

On the other hand, logic programming is also an important knowledge representation paradigm, with several implemented systems (e.g., XSB [26] and variants of Prolog) and a large user community. This community is advocating to extend the current Semantic Web with means to specify and exchange Logic Programming-style rules. However, up until now, these two formalisms have been largely separate.

To achieve a sensible architecture of the Semantic Web, rules and ontologies should not be unrelated and separate from each other. Hence, an integration of both techniques to data modelling is required. This deliverable is mainly concerned with bridging this gap.

Contribution We develop the Description Logic Programs (DLP) paradigm [12] to address this issue and establish a degree of interoperability by defining a meaning preserving translation between Description Logics ontologies and logic programs. Essentially we allow a commonly used subset of OWL to be fully captured in logic programs. To this extent, we study some of the more expressive features of OWL, such as nominals and functional restrictions, and show how these can be translated into logic programs. Finally, we present a prototypical implementation that shows in practise how a logic programming system can reason with a DL ontology. The feasibility of doing so is underlined in the first evaluation that we conducted.

While our translation basically targets Datalog-style rules, we show how to capture more expressive DL primitives, such as existentials using an Datalog extended with function symbols. This enables the modelling of incomplete information, allowing a distinction to be made between stating that someone has a son (whose name is not known), that someone has a son named “John”, or that it is unknown if they have a son or not. Similarly, nominals allow concepts to be defined by referring to well-known instances. Finally, functional properties allow uniqueness conditions to be captured, which is particularly important for interfacing with relational databases, where primary keys are used to uniquely identify records.

Non-Contribution This report is not concerned with a particular Web syntax for the rule layer, we believe that the issue of designing an RDF/XML-based syntax for Datalog is straightforward and does not impose any research questions. Wherever we present rules, we use the established LP-style syntax to improve the readability.

Related Work Our work conceptually follows from the relationship between description logic and first-order logic ([4]) and the multi-modal logic **K** ([27]). Hereby, we differ from other approaches who try to provide rules on top of ontologies.

Several approaches try to provide an axiomatisation of OWL or its precursor DAML+OIL, for example [10] provide an axiomatisation in Knowledge Interchange Format (KIF). This approach allows to specify further axioms (or rules) that could stem from a rule syntax,

but is - due to the features available in KIF - not directly executable using standard logic programming systems [31]. Two other systems try to implement OWL using a rule-based formalism: The Euler Proof Mechanism [25] by Jos De Roo of Agfa and Tim Berners-Lee's Closed World Machine (CWM) [3]. Both approaches axiomatise even elementary logic constructs. Their axiomatisation is neither proven to be sound nor complete, e.g. the substitutivity of equality is not correctly captured. There is no formal characterization of the inference algorithms that these systems employ. In our approach, however, we rely on the well-known semantics and evaluation procedures of Horn logic and extensively reuse existing constructs for our purposes.

Finally, several systems, such as CARIN [20] or AL-log [8], attempted the integration of description logic with Datalog-style rules. These systems do not provide a translation of one formalism into another as we do it. Rather, they investigate which primitives from both formalisms can be successfully combined, resulting in a decidable system.

Structure of the paper The remainder of this report is structured as follows: Section 2 gives an overview of our approach. Section 3 recapitulates the fundamentals of Description Logics and Logic Programming. Section 4 details the translation of DL-based ontologies into logic programs, which is central to our approach. Section 5 presents our prototypical implementation and further possibilities to integrate data. Section 6 reports the results of a first evaluation of our approach. We conclude in Section 7 summarizing our findings and giving an outlook to future work.

2 Overview

This section gives an overview of our approach and sketches the outline of the remainder of the paper. Our approach is driven by the insight that understanding the expressive *intersection* of the two KRs will be crucial to understanding the expressive *combination/union* of the two KRs. Hence, we start with the goal of understanding the relationship between both logic based KR formalisms (so as to be able to combine knowledge taken from both): Description Logics (decidable fragments of FOL closely related to propositional modal and dynamic logics [27]), and Logic Programs (see, e.g., [1] for review) which in turn is closely related to the Horn fragment of FOL. Since Description Logics resemble a subset of FOL without function symbols, we similarly focus on the fragment of Horn FOL, *def-Horn* , that contains no function symbols. Both DL and LP are then related to def-Horn.

The established correspondence is used to define a new intermediate KR called *Description Horn Logic* (DHL), which is contained within the intersection, and the closely related *Description Logic Programs* (DLP), which can be viewed as DHL with a moderate weakening as to the kinds of conclusion can be drawn.

Figure 1 illustrates the relationship between the various KRs and their expressive classes. DL and Horn are strict (decidable) subsets of FOL. LP, on the other hand, intersects with FOL but neither includes nor is fully included by FOL. For example FOL can express (positive) disjunctions, which are inexpressible in LP. On the other hand, several expressive features of LP, which are frequently used in practical rule-based applications,

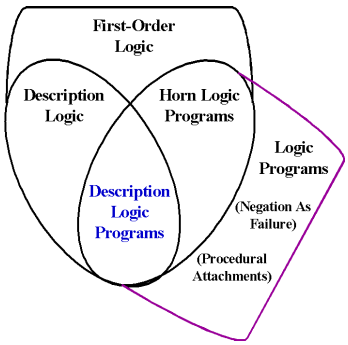

Figure 1: Expressive overlap of DL with LP.

are inexpressible in FOL (and consequently also outside of def-Horn). One example is negation-as-failure, a basic kind of logical non-monotonicity. Another example is procedural attachments, e.g., the association of action-performing procedural invocations with the drawing of conclusions about particular predicates.

Description Logic Programs, our newly defined intermediate KR, is contained within the intersection of DL and LP. "Full" LP, including non-monotonicity and procedural attachments, can thus be viewed as including an "ontology sub-language", namely the DLP subset of DL.

Rather than working from the intersection as we do in this paper, one may instead directly address the expressive union of DL and LP by studying the expressive union of DL and LP within the overall framework of FOL. This is certainly an interesting thing to do. However, to our knowledge, this has not yet been well characterised theoretically, e.g., it is unclear how, if at all, such a union differs from full FOL.

Full FOL has some significant practical and expressive drawbacks as a KR in which to combine DL and rules. First, full FOL has severe computational complexity: it is undecidable in the general case, and intractable even under the Datalog restriction (see Section 3). Second, it is not understood even at a basic research level how to expressively extend full FOL to provide non-monotonicity and procedural attachments; yet these are crucial expressive features in many (perhaps most) practical usages of rules. Third, full FOL and its inferencing techniques have severe practicable limitations since it is unfamiliar to the great majority of mainstream software engineers, whereas rules (e.g., in the form of SQL-type queries, or Prolog) are familiar conceptually to many of them.

Via the DLP KR, we give a new technique to combine DL and LP. We show how to perform *DLP-fusion*: the bidirectional mapping of premises and inferences (including typical kinds of queries) from the DLP fragment of DL to LP, and from the DLP fragment of LP to DL. DLP-fusion allows us to fuse the two logical KRs so that information from each can be used in the other. The DLP-fusion technique promises several benefits. In particular, DLP-fusion enables one to "build rules on top of ontologies": it enables the rule KR to have access to DL ontological definitions for vocabulary primitives (e.g., predicates and individual constants) used by the rules. Conversely, the technique enables one to "build ontologies on top of rules": it enables ontological definitions to be supplemented by rules, or imported into DL from rules. It also enables efficient LP inferencing algorithms/implementations, e.g., rule or relational database engines, to be exploited for

reasoning over large-scale DL ontologies.

3 Fundamentals

In this section be give a brief recapitulation of the formalisms we deal with, in particular, description logics and logic programming.

3.1 Description Logics

Among the multitude of available description logics, we chose to base our translation on the variant taken as the basis for the Web Ontology Language (OWL) [22]. OWL is expected to be a central standard for ontologies on the Semantic Web and corresponds to a very expressive description logic known as $\mathcal{SHOIN}(\mathbf{D}^+)$ [18, 16]. However, our results are not restricted to this description logic only.

DL	FOL
$\top$	$true.$
$a:C$	$C(a)$
$\langle a,b\rangle : P$	$P(a,b)$
$C \sqsubseteq D$	$\forall x.C(x) \rightarrow D(x)$
$P^+ \sqsubseteq P$	$\forall x,y,z.(P(x,y) \wedge P(y,z)) \rightarrow P(x,z)$
$C_1 \sqcap \ldots \sqcap C_n$	$C_1(x) \wedge \ldots \wedge C_n(x)$
$P \equiv Q^-$	$\forall x,y.P(x,y) \iff Q(y,x)$
$\exists P.C$	$\exists y.(P(x,y) \wedge C(y))$
$\forall P.C$	$\forall y.(P(x,y) \rightarrow C(y))$
$\top \sqsubseteq \leqslant 1P$	$\forall x,y,z.(P(x,y) \wedge P(x,z)) \rightarrow y = z$
$\{a_1,\ldots,a_n\}$	$x = a_1 \vee \ldots \vee x = a_n$
$C_1 \sqcup \ldots \sqcup C_n$	$C_1(x) \vee \ldots \vee C_n(x)$
$\neg C$	$\neg C(x)$
$\geqslant nP$	$\exists y_1,\ldots,y_n.\bigwedge_{1\leqslant i\leqslant n}(P(x,y_i)$ $\wedge \bigwedge_{1\leqslant i<j\leqslant n}(y_i \neq y_j)$
$\leqslant (n-1)P$	$\forall y_1,\ldots,y_n.(\bigwedge_{1\leqslant i\leqslant n}(P(x,y_i)))$ $\rightarrow \bigvee_{1\leqslant i<j\leqslant n}(y_i = y_j)$

Table 1: Syntax and Semantics of Description Logics

OWL Syntactic Shortcuts
disjointWith:
$C_1 \sqsubseteq \neg C_2$
differentIndividualFrom:
$\{i_1\} \sqsubseteq \neg\{i_2\}$
SymmetricProperty:
$P \equiv P^-$
FunctionalProperty:
$\top \sqsubseteq \leqslant 1P$
InverseFunctionalProperty:
$\top \sqsubseteq \leqslant 1P^-$
domain:
$\exists P.\top \sqsubseteq C$
range:
$\top \sqsubseteq \forall P.C$

Table 2: Syntactic Shortcuts

Syntactically, description logic concepts are defined by means of simple concept expressions, were complex concept expressions can be built from simpler ones by combining them using various connectives, e.g. conjunction and union. Concept expressions can be used to state various types of axioms, e.g. subset or equivalence relationships between concepts. Although the semantics of a description logic is usually given denotationally, in this paper we focus on the semantics established through correspondence with first order logic. In particular, the semantics of concept expressions is given by FOL formulas with one free variable, whereas the semantics of axioms is given by closed FOL formulas. The syntax and semantics of description logics is given in Table 1 and is based on [4].

OWL introduces several syntactic constructs that are convenience abbreviations of other $\mathcal{SHOIN}(\mathbf{D}^+)$ constructs. Without loss of generality we assume that all such shortcuts have previously been eliminated according to the expansions in Table 2.

3.2 Logic Programming

Logic programming is a knowledge representation mechanism based on Horn clauses – FOL implications with only one literal in the implication head and conjunctions of literals in the body, with all variables universally quantified. Horn clauses are often extended with a closed-world negation and arithmetic predicates, thus resulting in a classical logic programming environment.

The elementary syntactic units of the langue are terms, which can either be constant symbols such as names (so-called *atomic values*) and numbers (integer, float, etc.), variable symbols (usually written in capital letters) or compound terms of the form $f(t_1,\ldots,t_n)$ where t_i are terms, representing an invocation of the function f with arguments $t_1,\ldots,t_n$. Terms are used to form literals of the form $p(t_1,\ldots,t_n)$, where p is a predicate symbol, meaning that p holds for $t_1,\ldots,t_n$. Literals can be assembled in rules (also called *clauses* or implications) of the form H :- $B_1,\ldots,B_n$, which intuitively mean that the head literal H is true if all body literals B_i are all true. For example, the following rule expresses the fact that uncles of a person are the brothers of a parent:

$$\text{uncleOf}(X,Y) \text{ :- } \text{hasParent}(X,Z), \text{brotherOf}(Z,Y).$$

Rules with empty bodies are called *facts* – they define things that are always true. Rules in the program may be *recursive*, meaning that some predicates occur in the head of some and in the body of some other rules. Further, rules typically must be safe, meaning that every variable in the rule must appear in at least one positive literal.

Queries are conjunctions of literals of the form $Q_1,\ldots,Q_n$. For example, $\text{uncleOf}(X,Z)$ represents a query to retrieve all pairs where the first object is an uncle of the second one. Evaluating a query in a logic program means determining all variable substitutions for which the query conjuncts are all true.

Such declarative logic programs (LP) is the KR whose semantics underlies in a large part the four families of rule systems that are currently most commercially important—SQL relational databases, OPS5-heritage production rules, Prolog, and Event-Condition-Action rules—as well as the proposals for rules in context of the Semantic Web.

As mentioned before, the commonly used expressiveness of full LP includes features, notably negation-as-failure/priorities and procedural attachments, that are not expressible in FOL, much less in DL. We thus concentrate on only an expressive portion of LP, definite LP.

A *definite* LP is an ordinary LP in which negation-as-failure does not appear, i.e., a set of rules each having the form:

$H \leftarrow B_1 \wedge \ldots \wedge B_m$

where H, B_i are atoms, and $m \geq 0$.

Definite LP is closely related syntactically and semantically to the Horn fragment of FOL, a.k.a. Horn-clause logic. A clause in FOL has the form:

$L_1 \vee \ldots \vee L_k$

where each L_i is a (classical) literal. A literal L has either the form (1) A or (2) $\neg A$, where A is an atom. The literal is said to be *positive* in case (1), or to be *negative* in case (2). A clause is said to be *Horn* when *at most one* of its literals is positive. A Horn clause is said to be *definite* when *exactly one* of its literals is positive. A definite Horn clause is also known as a *Horn rule*. A definite Horn clause, a.k.a. *Horn rule*, can thus be written in the form:

$H \leftarrow B_1 \wedge \ldots \wedge B_m$

where H, B_i are atoms, and $m \geq 0$. We say that this Horn rule *corresponds* to the definite LP rule that has the same syntactic form, and vice versa. Likewise, we say that a Horn ruleset $\mathcal{RH}$ and a definite LP ruleset $\mathcal{RP}$ correspond to each other when their rules do (isomorphically). We then also say that $\mathcal{RP}$ is the *LP-correspondent* of $\mathcal{RH}$, and conversely that $\mathcal{RH}$ is the *Horn-correspondent* of $\mathcal{RP}$.

As mentioned above, it is implicit in this notation that all logical variables are universally quantified at the outer level, i.e., over the scope of the whole clause. E.g., the rule

$man(x) \leftarrow human(x) \wedge male(x)$

can be written equivalently as:

$\forall x.\, man(x) \leftarrow human(x) \wedge male(x).$

Note the similarity with the FOL equivalent of a DL inclusion (subClassOf) axiom given in Table 1.

An LP rule or Horn clause is said to be *equality-free* when the equality predicate does not appear in it. Likewise, each is said to be *Datalog* when no logical functions (of arity greater than zero) appear in it. The Datalog restriction is usually taken to mean, additionally, no negation and only "safe" rules (all variables occuring in the head of a rule also occur in the body of a rule).

The semantics of an ordinary LP is defined to be a *conclusion set*, where each conclusion is a ground atom, i.e., fact, entailed by the LP. Formally, the semantics of a definite LP $\mathcal{R}$ is defined as follows. Let HB stand for the Herbrand base of $\mathcal{R}$. The conclusion set $\mathcal{C}$ is the smallest (w.r.t. set inclusion) subset $\mathcal{S}$ of HB such that for any rule

$H \leftarrow B_1 \wedge \ldots \wedge B_m$,

if $B_1 \wedge \ldots \wedge B_m \in \mathcal{S}$ then $H \in \mathcal{S}$.

The relationship of LP semantics to FOL semantics is relatively simple to describe for the case of definite equality-free Datalog LP, which we call *def-LP*. The syntactically corresponding fragment of FOL is definite equality-free Datalog Horn FOL, which we call *def-Horn*. Let $\mathcal{RP}$ be a def-LP. Let $\mathcal{RH}$ stand for the corresponding def-Horn ruleset. The conclusion set of $\mathcal{RP}$ then coincides with the smallest (w.r.t. set inclusion) Herbrand model of $\mathcal{RH}$.

Hence, the def-LP and the def-Horn ruleset entail exactly the same set of facts. Every conclusion of the def-LP is also a conclusion of the def-Horn ruleset. Relative to the def-Horn ruleset, the def-LP is thus sound; moreover, it is complete for fact-form conclusions, i.e., for queries whose answers amount to conjunctions of facts. However, the def-LP is a mildly *weaker* version of the def-Horn ruleset, in the following sense. Every conclusion of the def-LP must have the form of a fact. By contrast, the entailments, i.e., conclusions, of the def-Horn ruleset are not restricted to be facts. E.g., suppose $\mathcal{RH}$ consists of the two rules

$kiteDay(Tues) \leftarrow sunny(Tues) \wedge windy(Tues)$

and

$sunny(Tues)$.

Then it entails

$kiteDay(Tues) \leftarrow windy(Tues)$

(a non-unit derived clause) whereas $\mathcal{RP}$ does not. In practical applications, however, quite often only the fact-form conclusions are desired, e.g., an application might be interested above only in whether or not $kiteDay(Tues)$ is entailed. The def-LP has the virtue of conceptual and computational simplicity. Thinking in terms of expressive classes, we will view def-LP as an *expressive subset* of def-Horn—we will call it the expressive *f-subset*. def-LP is a mild weakening of def-Horn along the dimension of entailment power, permitting only fact-form conclusions—we will call this *f-weakening*.

In return for this f-weakening, def-LP has some quite attractive computational characteristics (as well as being expressively extensible in directions that FOL is not, as discussed earlier). For the propositional case of def-LP, exhaustive inferencing is $O(n)$ where $n = |\mathcal{RP}|$ — i.e., worst-case linear time [9]. For the general case with logical variables, the entire conclusion set of a def-LP $\mathcal{RP}$ can be computed in time $O(n^{v+1})$, when there is a constant bound v on the number of logical variables per rule (this restriction, which we will call *VB*, is typically met in practise). Inferencing in def-LP is thus tractable (worst-case polynomial time) given VB. In contrast, DLs are generally not tractable (typically ExpTime or even NExpTime complexity for key inference problems), and full FOL is not decidable.

4 Translation

In this section we will discuss how DL languages (e.g., OWL) can be translated to def-Horn, and vice versa.

4.1 Expressive Restrictions

We will first discuss the expressive restrictions of DL and def-Horn as these will constrain the subset of DL and def-Horn for which a complete mapping can be defined.

Tree model property DLs are decidable subsets of FOL where the decidability is due in large part to their having (a form of) the tree model property [30]. Expressive features such as transitive properties and the `oneOf` constructor compromise the tree model property to some extent, e.g., transitive properties can cause "short-cuts" down branches of the tree. This property says that a DL class C has a model (an interpretation I in which C^I is non-empty) iff C has a tree-shaped model, i.e., one in which the interpretation of properties defines a tree shaped directed graph. This requirement severely restricts the way variables and quantifiers can be used. In particular, quantifiers must be *relativised* via atomic formulae (as in the guarded fragment of FOL [11]), i.e., the quantified variable must occur in a property predicate along with the free variable (recall that DL classes correspond to formulae with one free variable). For example, the DL class $\exists P.C$ corresponds to the FOL formula $\exists y.(P(x,y) \wedge C(y))$, where the property predicate P acts as a guard.

One obvious consequence of this restriction is that it is impossible to describe classes whose instances are related to another anonymous individual via different property paths. For example, it is impossible to assert that individuals who live and work at the same location are "HomeWorkers". This is easy with a Horn rule, e.g.:

$$\text{HomeWorker}(x) \leftarrow \text{work}(x,y) \wedge \text{live}(x,z) \wedge \text{loc}(y,w) \wedge \text{loc}(z,w)$$

Arity of predicates Another restriction in DLs is that only unary and binary predicates can usually be captured. This is not an inherent restriction, and n-ary DLs are known, e.g., $\mathcal{DLR}$ [7]. This is a less onerous restriction, however, as techniques for reifying higher arity predicates are well known [17].

Quantifiers Definite Horn FOL requires that all variables are universally quantified (at the outer level of the rule), and restricts logical connectives in certain ways. One obvious consequence of the restriction on quantifiers is that it is impossible to assert the existence of individuals whose identity might not be known. For example, it is impossible to assert that all persons have a father (known or unknown). This is easy with a DL axiom, e.g.:

$$\text{Person} \sqsubseteq \exists \text{father}.\top.$$

Logic Programming systems typically deal with existential quantification via a procedure called skolemization. Skolemization is a syntactic trnasformation routinely used in automatic inference systems in which existential variables are replaced by 'new' functions applied to univerisal variables in front of the existential quantifier of the variable. This procedure is correct with respect to satifisfiability, that is if the original formula is satisfiable, then so is the skolemized formula, for example a FOL formula

$$\forall x \exists y p(x,y)$$

which is not directly expressible in Horn logic, would be translated to

$$\forall x p(x, f(x))$$

f has to be a previously unused function symbol, otherwise unintentional equalities could follow.

Negation The reason for this is that negation in description logics and logic programming has significantly different semantics. In description logics negation has so-called classical semantics, which is quite different from negation-as-failure, typically employed in logic programming. In a logic program something is false if it can't be proven to be true. E.g., unless the database contains Composer(johann-sebastian), one can entail $\neg$Composer(johann-sebastian). On the other hand, in classical logic, the lack of information that Johann Sebastian Bach is a composer doesn't allow concluding that he is not a composer. This is related to the closed-world assumption: a logic program assumes that it knows all relevant facts and everything else is assumed to be false. A description logic ontology, on the other hand, assumes it doesn't know all the facts, so it must be explicitly told which facts are false.

Hence, no negation may appear within the body of a rule, nor within the head. Following the De Morgan laws, since has the direct consequences that neither existentials nor disjunctions may appear within the head of rules. Thus it is impossible to assert, e.g., that all men or woman are persons (and no one can be at the same time be a man and a woman). This would also be easy using DL axioms, e.g.:

$$\begin{aligned} \text{Man} \sqcup \text{Woman} &\sqsubseteq \text{Person} \\ \text{Man} &\sqsubseteq \neg\text{Woman}. \end{aligned}$$

4.2 Mapping Statements

In this section, we show how (some of) the *statements* (axioms) of DL and DL based languages (such as DAML+OIL and OWL) correspond to def-Horn statements (rules).

4.2.1 RDFS Statements

RDFS provides a subset of the DL statements described in Section 3.1: subclass, subproperty, range, and domain statements (which in a DL setting are often called Tbox axioms); and asserted class-instance (type) and instance-property-instance relationships (which in a DL setting are often called Abox axioms).

As we saw in Section 3.1, a DL inclusion axiom corresponds to an FOL implication. This leads to a straightforward mapping from class and property inclusion axioms to def-Horn rules as follows:

$C \sqsubseteq D$, i.e., class C is subclass of class D, maps to:
$$D(x) \leftarrow C(x)$$

$Q \sqsubseteq P$, i.e., Q is a subproperty of P, maps to:
$$P(x,y) \leftarrow Q(x,y)$$

As shown in Table 1, RDFS range and domain statements correspond to DL axioms of the form $\top \sqsubseteq \forall P.C$ (range of P is C) and $\top \sqsubseteq \forall P^{-}.C$ (domain of P is C). From Table 1, we can see that these are equivalent to the FOL sentences $\forall x.\ true \rightarrow (\forall y.\ P(x,y) \rightarrow C(y))$ and $\forall x.\ true \rightarrow (\forall y.\ P(y,x) \rightarrow C(y))$, which can be simplified to $\forall x,y.\ P(x,y) \rightarrow C(y)$ and $\forall x,y.\ P(y,x) \rightarrow C(y)$ respectively. These FOL sentences are already in def-Horn form, which gives us the following mappings for range and domain:

$\top \sqsubseteq \forall P.C$, i.e., the range of property P is class C, maps to:
$$C(y) \leftarrow P(x,y)$$

$\top \sqsubseteq \forall P^{-}.C$, i.e., the domain of property P is class C, maps to:
$$C(y) \leftarrow P(y,x)$$

Finally, asserted class-instance (type) and instance-property-instance relationships, which correspond to DL axioms of the form $a : C$ and $\langle a,b \rangle : P$ respectively (Abox axioms), are equivalent to FOL sentences of the form $C(a)$ and $P(a,b)$, where a and b are constants. These are already in def-Horn form: they are simply rules with empty bodies (which are normally omitted):

$a : C$, i.e., the individual a is an instance of the class C, maps to:

$$C(a)$$

$\langle a, b \rangle : P$, i.e., the individual a is related to the individual b via the property P, maps to:

$$P(a, b)$$

Note that in these rules a and b are ground (constants).

4.2.2 OWL statements

OWL extends RDF with additional statements about classes and properties (Tbox axioms). In particular, it adds explicit statements about class, property and individual equality and inequality, as well as statements asserting property inverses, transitivity, functionality (unique) and inverse functionality (unambiguous).

As discussed in Section 3.1, class and property equivalence axioms can be replaced with a symmetrical pair of inclusion axioms, so they can be mapped to a symmetrical pair of def-Horn rules as follows:

$C \equiv D$, i.e., the class C is equivalent to (has the same extension as) the class D, maps to:

$$\begin{array}{rcl} D(x) & \leftarrow & C(x) \\ C(x) & \leftarrow & D(x) \end{array}$$

$P \equiv Q$, i.e., the property P is equivalent to (has the same extension as) the property Q, maps to:

$$\begin{array}{rcl} Q(x,y) & \leftarrow & P(x,y) \\ P(x,y) & \leftarrow & Q(x,y) \end{array}$$

As we saw in Section 3.1, the semantics of inverse axioms of the form $P \equiv Q^-$ are captured by FOL sentences of the form $\forall x,y.P(x,y) \iff Q(x,y)$, and the semantics of transitivity axioms of the form $P^+ \sqsubseteq P$ are captured by FOL sentences of the form $\forall x,y,z.P(x,y) \wedge P(y,z) \rightarrow P(x,z)$. This leads to a direct mapping into def-Horn as follows:

$P \equiv Q^-$, i.e., the property Q is the inverse of the property P, maps to:

$$\begin{array}{rcl} Q(y,x) & \leftarrow & P(x,y) \\ P(x,y) & \leftarrow & Q(y,x) \end{array}$$

$P^+ \sqsubseteq P$, i.e., the property P is transitive, maps to:

$$P(x,z) \leftarrow P(x,y) \wedge P(y,z)$$

As we saw in Section 3.1, DL axioms asserting the functionality of properties correspond to FOL sentences with equality. E.g., a DL axiom $\top \sqsubseteq \leqslant 1P$ (P is a functional property) corresponds to the FOL sentence $\forall x,y,z.P(x,y) \wedge P(x,z) \rightarrow y = z$.Note that, technically, this is partial-functionality as for any given x there is no requirement that there exist a y such that $P(x,y)$. This kind of axiom cannot be dealt with in our current framework (see Section 4.1) as it would require def-Horn rules with equality in the head, i.e., rules of the form $(y = z) \leftarrow P(x,y) \wedge P(x,z)$.

4.3 Mapping Class Constructors

In the previous section we showed how DL axioms correspond with def-Horn rules, and how these can be used to make statements about classes and properties. In DLs, the classes appearing in such axioms need not be atomic, but can be complex compound expressions built up from atomic classes and properties using a variety of constructors. A great deal of the power of DLs derives from this feature, and in particular from the set of constructors provided.Note that this feature is not supported in the RDFS subset of DLs. In the following section we will show how these DL expressions correspond to expressions in the body of def-Horn rules.

In the following we will, as usual, use C, D to denote classes, P, Q to denote properties and n to denote an integer.

Conjunction (DL $\sqcap$)

A DL class can be formed by conjoining existing classes, e.g., $C \sqcap D$. From Table 1 it can be seen that this corresponds to a conjunction of unary predicates. Conjunction can be directly expressed in the body of a def-Horn rule. E.g., when a conjunction occurs on the l.h.s. of a subclass axiom, it simply becomes conjunction in the body of the corresponding rule

$$C_1 \sqcap C_2 \sqsubseteq D \quad \equiv \quad D(x) \;\leftarrow\; C_1(x) \wedge C_2(x)$$

Similarly, when a conjunction occurs on the r.h.s. of a subclass axiom, it becomes conjunction in the head of the corresponding rule:

$$C \sqsubseteq D_1 \sqcap D_2 \quad \equiv \quad D_1(x) \wedge D_2(x) \;\leftarrow\; C(x),$$

This is then easily transformed (via the Lloyd-Topor transformations [21]) into a pair of def-Horn rules:

$$\begin{aligned} D_1(x) &\leftarrow C(x) \\ D_2(x) &\leftarrow C(x) \end{aligned}$$

Disjunction (DL $\sqcup$)

A DL class can be formed from a disjunction of existing classes, e.g., $C \sqcup D$. From Table 1 it can be seen that this corresponds to a disjunction of unary predicates. When a disjunction occurs on the l.h.s. of a subclass axiom it simply becomes disjunction in the body of the corresponding rule:

$$C_1 \sqcup C_2 \sqsubseteq D \quad \equiv \quad D(x) \;\leftarrow\; C_1(x) \vee C_2(x)$$

This is easily transformed (again by Lloyd-Topor) into a pair of def-Horn rules:

$$\begin{aligned} D(x) &\leftarrow C_1(x) \\ D(x) &\leftarrow C_2(x) \end{aligned}$$

When a disjunction occurs on the r.h.s. of a subclass axiom it becomes a disjunction in the head of the corresponding rule, and this cannot be handled within the def-Horn framework.

Universal Restriction (DL $\forall$)

In a DL, the universal quantifier can only be used in *restrictions*—expressions of the form $\forall P.C$ (see Section 4.1). This is equivalent to an FOL clause of the form $\forall y.P(x,y) \rightarrow C(y)$ (see Table 1). P must be a single primitive property, but C may be a compound expression. Therefore, when a universal restriction occurs on the r.h.s. of a subclass axiom it becomes an implication in the head of the corresponding rule:

$$C \sqsubseteq \forall P.D \quad \equiv \quad (D(y) \ \leftarrow \ P(x,y)) \ \leftarrow \ C(x),$$

which is easily transformed into the standard def-Horn rule:

$$D(y) \ \leftarrow \ C(x) \wedge P(x,y).$$

When a universal restriction occurs on the l.h.s. of a subclass axiom it becomes an implication in the body of the corresponding rule. This cannot, in general, be mapped into def-Horn as it would require negation in a rule body.

Existential Restriction (DL $\exists$)

In a DL, the existential quantifier (like the universal quantifier) can only be used in restrictions of the form $\exists P.C$. This is equivalent to an FOL clause of the form $\exists y.P(x,y) \wedge C(y)$ (see Table 1). P must be a single primitive property, but C may be a compound expression.

When an existential restriction occurs on the l.h.s. of a subclass axiom, it becomes a conjunction in the body of a standard def-Horn rule:

$$\exists P.C \sqsubseteq D \quad \equiv \quad D(x) \ \leftarrow \ P(x,y) \wedge C(y).$$

When an existential restriction occurs on the r.h.s. of a subclass axiom, it becomes a conjunction in the head of the corresponding rule, with a variable that is existentially quantified. This cannot be handled within the def-Horn framework.

Negation and Cardinality Restrictions (DL $\neg$, $\geqslant$ and $\leqslant$)

These constructors cannot, in general, be mapped into def-Horn. The case of negation is obvious as negation is not allowed in either the head or body of a def-Horn rule. As can be seen in Table 1, cardinality restrictions correspond to assertions of variable equality and inequality in FOL, and this is again outside of the def-Horn framework.

In some cases, however, it would be possible to simplify the DL expression using the usual rewriting tautologies of FOL in order to eliminate the offending operator(s). For example, negation can always be pushed inward by using a combination of De Morgan's laws and equivalences such as $\neg\exists P.C \equiv \forall P.\neg C$ and $\neg \geqslant n\,P \equiv \leqslant (n-1)\,P$. Further simplifications are also possible, e.g., using the equivalences $C \sqcup \neg C \equiv \top$, and $\forall P.\top \equiv \top$. For the sake of simplicity, however, we will assume that DL expressions are in a canonical form where all relevant simplifications have been carried out.

4.4 Defining DHL via a Recursive Mapping from DL to def-Horn

As we saw in Section 4.2, some DL constructors (conjunction and universal restriction) can be mapped to the heads of rules whenever they occur on the r.h.s. of an inclusion axiom, while some DL constructors (conjunction, disjunction and existential restriction) can be mapped to the bodies of rules whenever they occur on the l.h.s. of an inclusion axiom. This naturally leads to the definition of two DL languages, classes from which can be mapped into the head or body of LP rules; we will refer to these two languages as $\mathcal{L}_h$ and $\mathcal{L}_b$ respectively.

The syntax of the two languages is defined as follows. In both languages an atomic name A is a class, and if C and D are classes, then $C \sqcap D$ is also a class. In $\mathcal{L}_h$, if C is a class and R is a property, then $\forall R.C$ is also a class, while in $\mathcal{L}_b$, if D, C are classes and R is a property, then $C \sqcup D$ and $\exists R.C$ are also classes.

Using the mappings from Section 4.2, we can now follow the approach of [4] and define a recursive mapping function $\mathcal{T}$ which takes a DL axiom of the form $C \sqsubseteq D$, where C is an $\mathcal{L}_b$-class and D is an $\mathcal{L}_h$-class, and maps it into an LP rule of the form $A \leftarrow B$. The mapping is defined as follows:

$$\begin{array}{lcl}
\mathcal{T}(C \sqsubseteq D) & \longrightarrow & Th(D,y) \leftarrow Tb(C,y) \\
Th(A,x) & \longrightarrow & A(x) \\
Th((C \sqcap D),x) & \longrightarrow & Th(C,x) \wedge Th(D,x) \\
Th((\forall R.C),x) & \longrightarrow & Th(C,y) \leftarrow R(x,y) \\
Tb(A,x) & \longrightarrow & A(x) \\
Tb((C \sqcap D),x) & \longrightarrow & Tb(C,x) \wedge Tb(D,x) \\
Tb((C \sqcup D),x) & \longrightarrow & Tb(C,x) \vee Tb(D,x) \\
Tb((\exists R.C),x) & \longrightarrow & R(x,y) \wedge Tb(C,y)
\end{array}$$

where A is an atomic class name, C and D are classes, R is a property and x, y are variables, with y being a "fresh" variable, i.e., one that has not previously been used.

As we saw in Section 4.2, rules of the form $(H \wedge H') \leftarrow B$ are rewritten as two rules $H \leftarrow B$ and $H' \leftarrow B$; rules of the form $(H \leftarrow H') \leftarrow B$ are rewritten as $H \leftarrow (B \wedge H')$; and rules of the form $H \leftarrow (B \vee B')$ are rewritten as two rules $H \leftarrow B$ and $H \leftarrow B'$.

For example, $\mathcal{T}$ would map the DL axiom

$$A \sqcap \exists R.C \sqsubseteq B \sqcap \forall P.D$$

into the LP rule

$$B(x) \wedge (D(z) \leftarrow P(x,z)) \leftarrow A(x) \wedge R(x,y) \wedge C(x)$$

which is rewritten as the pair of rules

$$\begin{array}{lcl}
B(x) & \leftarrow & A(x) \wedge R(x,y) \wedge C(x) \\
D(z) & \leftarrow & A(x) \wedge R(x,y) \wedge C(x) \wedge P(x,z).
\end{array}$$

We call $\mathcal{L}$ the intersection of $\mathcal{L}_h$ and $\mathcal{L}_b$, i.e., the language where an atomic name A is a class, and if C and D are classes, then $C \sqcap D$ is also a class. We then extend $\mathcal{T}$ to deal

with axioms of the form $C \equiv D$, where C and D are both $\mathcal{L}$-classes:

$$\mathcal{T}(C \equiv D) \longrightarrow \begin{cases} \mathcal{T}(C \sqsubseteq D) \\ \mathcal{T}(D \sqsubseteq C) \end{cases}$$

As we saw in Section 4.2.1, range and domain axioms $\top \sqsubseteq \forall P.D$ and $\top \sqsubseteq \forall P^{-}.D$ are mapped into def-Horn rules of the form $D(y) \leftarrow P(x,y)$ and $D(x) \leftarrow P(x,y)$ respectively. Moreover, class-instance and instance-property-instance axioms $a : D$ and $\langle a,b \rangle : P$ are mapped into def-Horn facts (i.e., rules with empty bodies) of the form $D(a)$ and $P(a,b)$ respectively. We therefore extend $\mathcal{T}$ to deal with these axioms in the case that D is an $\mathcal{L}_h$-class:

$$\begin{array}{lcll} \mathcal{T}(\top \sqsubseteq \forall P.D) & \longrightarrow & Th(D,y) & \leftarrow P(x,y) \\ \mathcal{T}(\top \sqsubseteq \forall P^{-}.D) & \longrightarrow & Th(D,x) & \leftarrow P(x,y) \\ \mathcal{T}(a : D) & \longrightarrow & Th(D,a) & \\ \mathcal{T}(\langle a,b \rangle : P) & \longrightarrow & P(a,b) & \end{array}$$

where x,y are variables and a,b are constants.

Finally, we extend $\mathcal{T}$ to deal with the property axioms discussed in Section 4.2:

$$\begin{array}{lcl} \mathcal{T}(P \sqsubseteq Q) & \longrightarrow & Q(x,y) \leftarrow P(x,y) \\ \mathcal{T}(P \equiv Q) & \longrightarrow & \begin{cases} Q(x,y) \leftarrow P(x,y) \\ P(x,y) \leftarrow Q(x,y) \end{cases} \\ \mathcal{T}(P \equiv Q^{-}) & \longrightarrow & \begin{cases} Q(x,y) \leftarrow P(y,x) \\ P(y,x) \leftarrow Q(x,y) \end{cases} \\ \mathcal{T}(P^{+} \sqsubseteq P) & \longrightarrow & P(x,z) \leftarrow P(x,y) \wedge P(y,z) \end{array} \tag{1}$$

Definition 1 (Description Horn Logic) *A* Description Horn Logic (DHL) *ontology is a set of DHL axioms of the form $C \sqsubseteq D$, $A \equiv B$, $\top \sqsubseteq \forall P.D$, $\top \sqsubseteq \forall P^{-}.D$, $P \sqsubseteq Q$, $P \equiv Q$, $P \equiv Q^{-}$, $P^{+} \sqsubseteq P$, $a : D$ and $\langle a,b \rangle : P$, where C is an $\mathcal{L}_b$-class, D is an $\mathcal{L}_h$-class, A,B are $\mathcal{L}$-classes, P,Q are properties and a,b are individuals.*

Using the relationships of (full) DL to FOL discussed in Section 3.1, especially Table 1, it is straightforward to show the following.

Theorem 4.4.1 (Translation Semantics) *The mapping $\mathcal{T}$ preserves semantic equivalence. Let $\mathcal{K}$ be a DHL ontology and $\mathcal{H}$ be the def-Horn ruleset that results from applying the mapping $\mathcal{T}$ to all the axioms in $\mathcal{K}$. Then $\mathcal{H}$ is logically equivalent to $\mathcal{K}$ w.r.t. the semantics of FOL — $\mathcal{H}$ has the same set of models and entailed conclusions as $\mathcal{K}$.*

DHL can, therefore, be viewed alternatively and precisely as an expressive fragment of def-Horn— i.e., as the *range* of $\mathcal{T}(\text{DHL})$.

Definition 2 (Description Logic Programs) *We say that a def-LP $\mathcal{RP}$ is a* Description Logic Program (DLP) *when it is the LP-correspondent of some DHL ruleset $\mathcal{RH}$.*

A DLP is directly defined as the LP-correspondent of a def-Horn ruleset that results from applying the mapping $\mathcal{T}$. Semantically, a DLP is thus the f-weakening of that DHL ruleset (recall subsection 3.2). The DLP expressive class is thus the expressive f-subset

of DHL. By Theorem 4.4.1, DLP can, therefore, be viewed alternatively and precisely as an expressive subset of DL, not just of def-Horn.

In summary, expressively DLP is contained in DHL which in turn is contained in the expressive intersection of DL and Horn.

4.5 Expressive Power of DHL

Although the asymmetry of DHL (w.r.t. classes on different sides of axioms) makes it rather unusual by DL standards, it is easy to see that it includes (the OWL subset of) RDFS, as well as that part of OWL which corresponds to a simple frame language.

As far as RDFS is concerned, we saw in Section 4.2.1 that RDFS statements are equivalent to DL axioms of the form $C \sqsubseteq D$, $\top \sqsubseteq \forall P.C$, $\top \sqsubseteq \forall P^{-}.C$, $P \sqsubseteq Q$, $a : D$ and $\langle a, b \rangle : P$, where C, D are classes, P, Q are properties and a, b are individuals. Given that all RDFS classes are $\mathcal{L}$-classes (they are atomic class names), a set of DL axioms corresponding to RDFS statements would clearly satisfy the above definition of a DHL ontology.

DHL also includes the subset of OWL corresponding to simple frame language axioms, i.e., axioms defining a primitive hierarchy of classes, where each class is defined by a frame. A frame specifies the set of subsuming classes and a set of slot constraints. This corresponds very neatly to a set of DL axioms of the form $A \sqsubseteq \mathcal{L}_h$.

Moreover, DHL supports the extension of this language to include equivalence of conjunctions of atomic classes, and axioms corresponding to OWL transitive property, and inverse property statements.

4.6 Extending DHL

In this subsection we present the translations for $\top$, $\bot$, existential quantifiers, equality, nominals and functional properties which go beyond DHL, as they violate the Datalog restriction (where ever existential quantification is needs or translated rules are unsafe) or the equality-free restriction (functional properties). We consider these translations separately to keep DHL conceptually clean. However, the translations are practically relevant, since they can be implemented in a wide range of available systems, which implement the necessary functionality.

$\top$ and $\bot$. Concept expressions containing $\top$ and $\bot$ are translated into rules as follows:

$\mu[\top](X)$:- B	$\Rightarrow$	Delete the rule.
H :- $\mu[\top](X), B$	$\Rightarrow$	H :- B
$\mu[\bot](X)$:- B	$\Rightarrow$	:- B
H :- $\mu[\bot](X), B$	$\Rightarrow$	Delete the rule.

If B is empty, the second translation may result in an unsafe rule – variable X will occur in the rule head without occurring in the rule body. If that happens, the result of the operator is undefined. Further, the third translation results in a rule with an empty head. Such a rule is an integrity constraint – it specifies conditions which must not occur. Not all logic programming environments support integrity constraints, so in that case the result of μ is undefined.

Existential Quantifiers. Concept descriptions containing existential quantifiers are translated as follows (the symbol f must be a new, previously unused function symbol; variable Y is a new variable previously unused in the rule):

$\mu[\exists R.C](X)$:- B	$\Rightarrow$	$R(X, f(X))$:- B
	$\Rightarrow$	$\mu[C](f(X))$:- B
H :- $\mu[\exists R.C](X), B$	$\Rightarrow$	H :- $R(X,Y), \mu[C](Y), B$

The first translation is obtained by starting with the FOL encoding of the concept description, skolemising the existential quantifier in the head of the rule and splitting the conjunction in the head using the Lloyd-Topor transformation [21]:

$$\begin{gathered}\forall X.(\exists Y.(R(X,Y) \wedge C(Y)) \Leftarrow B) \Leftrightarrow \\ \forall X.((R(X, f(X)) \wedge C(f(X))) \Leftarrow B) \Leftrightarrow \\ \forall X.(R(X, f(X)) \Leftarrow B \wedge C(f(X)) \Leftarrow B)\end{gathered}$$

The second translation was already presented in [12] and is obtained also by starting with the FOL encoding of the concept description and simply moving the existential quantifier from the body of the rule outside:

$$\begin{gathered}\forall X.(H \Leftarrow \exists Y.(R(X,Y) \wedge C(Y)) \wedge B) \Leftrightarrow \\ \forall X \forall Y.(H \Leftarrow R(X,Y) \wedge C(Y) \wedge B)\end{gathered}$$

Instance Equivalence. Many translations of OWL descriptions require specifying that two instances are equivalent. However, this predicate is typically not available in logic programming environments, so we must provide one. As presented in [14], the correct semantics can be captured by five axioms of the equivalence. Reflexivity, symmetry and transitivity ensure that the predicate express the algebraic properties of equivalence. The substitutivity axioms ensure the correct semantics of equivalence for function and predicate symbols.

$$\begin{gathered}=(x,x). \\ =(x,y) \text{ :- } =(y,x). \\ =(x,z) \text{ :- } =(x,y) \wedge =(y,z). \\ =(f(x_1,..,x_n), f(y_1,..,y_n)) \text{ :- } \{=(x_i,y_i) | 1 \leqslant i \leqslant n\}. \\ Q(x_1,..,x_n) \text{ :- } Q(y_1,..,y_n) \cup \{=(x_i,y_i) | 1 \leqslant i \leqslant n\}.\end{gathered}$$

The reader may note that the axioms of substitutivity have to be instantiated for all predicates Q and functions f used in the rule base. Also, one may note that the reflexivity rule is unsafe, so one cannot execute it directly in a logic programming environment. We can provide a partial solution by explicitly instantiating a fact of the form $=(a,a)$ for all individuals in the program. However, this is only a partial solution – we'd have to do this for the set of all skolemised terms, but this set is unfortunately infinite. Further, many logic programming environments offer built-in predicates that can handle the reflexivity axiom efficiently.

Enumeration/Nominals. A naive translation of nominals $\{i_1,...,i_n\}$ would be to replace it with a new concept name N_i, along with membership assertions $N_i(i_j)$ for each nominal element. Such translation, however, is not correct, since the interpretation of nominals consists only of specified instances. Using Clark's completion, this can be written like this:

$$\forall X.(N_i(X) \Leftrightarrow X = i_1 \vee ... \vee X = i_n)$$

The naive translation axiomatises the Clark's completion right to left. To specify completeness of the set, however, one would have to interpret the rule left to right, which would require disjunction in the rule head. However, nominals of only one element can be handled, since only one disjunction in the head remains:

$\mu[\{i\}](X)$:- B	$\Rightarrow$	$N_i(X)$:- B
H :- $\mu[\{i\}](X), B$	$\Rightarrow$	H :- $N_i(X), B$
N_i is defined as: $N_i(i).$ $= (X,i)$:- $N_i(X).$		

This translation can further be optimized for the special case of the hasValue construct of OWL, where we can avoid introducing new predicate by simply replacing a variable with the individual:

$\mu[\exists R.\{i\}](X)$:- B	$\Rightarrow$	$R(X,i)$:- B
H :- $\mu[\exists R.\{i\}](X), B$	$\Rightarrow$	H :- $R(X,i), B$

Cardinality Statements. As may be observed from the Table 1, translation of cardinality constraints into first-order logic uses either disjunction in the head or the negation of the equivalence predicate. Horn logic systems don't provide these features, so cardinalities can't be handled directly.

However, the maximum cardinality restriction with the value of one in the head of the rule can be handled, since only one disjunct in the head of the rule remains. This is especially important since the unique and unambiguous property syntactic shortcuts (cf. Table 2) are expanded into such cardinality restrictions. The translation can be done as follows:

$\mu[\leqslant 1 R](X)$:- B	$\Rightarrow$	$= (Y,Z)$:- $R(X,Y), R(X,Z), B$

4.7 Translation Example

Our theoretical excursion illustrating the mapping of indivudal OWL language primitives is practically illustrated through the ontology presented in Table 3, describing some aspects about Johann-Sebastian Bach. The translation of the ontology into a logic program is shown in Table 4. To save space we don't repeat A-Box assertions which are syntactically identical to the assertions in the ontology. Also, we show only the substitutivity axioms for one function and one predicate symbol.

(T1)	$\text{Woman} \sqsubseteq \text{Person}$	(T7)	$\text{ancestorOf}^{+} \sqsubseteq \text{ancestorOf}$
(T2)	$\text{Man} \sqsubseteq \text{Person}$	(T8)	$\text{marriedTo}^{-} \sqsubseteq \text{marriedTo}$
(T3)	$\text{Wife} \sqsubseteq \text{Woman} \sqcap \exists \text{marriedTo.Husband}$	(T9)	$\exists \text{livesIn.}\{\text{leipzig}\} \sqsubseteq \text{LeipzigInhabitant}$
(T4)	$\text{Husband} \sqsubseteq \text{Man} \sqcap \exists \text{marriedTo.Wife}$	(T10)	$\text{Genius} \sqcap \text{Composer} \sqsubseteq$
(T5)	$\text{Father} \equiv \text{Man} \sqcap \exists \text{hasChild.Person}$		$\forall \text{hasComposed.Masterpiece}$
(T6)	$\text{hasChild} \sqsubseteq \text{ancestorOf}$		
(A1)	$\exists \text{hasChild.Man}(\text{johann-ambrosius})$	(A7)	$\text{Man}(\text{johann-ambrosius})$
(A2)	$\text{Composer} \sqcap \text{Man}(\text{johann-sebastian})$	(A8)	$\text{livesIn}(\text{johann-sebastian}, \text{lepzig})$
(A3)	$\text{Person}(\text{wilhelm-friedemann})$	(A9)	$\text{Genius}(\text{johann-sebastian})$
(A4)	$\text{hasChild}(\text{johann-sebastian},$ $\text{wilhelm-friedemann})$	(A10)	$\text{hasComposed}(\text{johann-sebastian},$ $\text{matthaeus-passion})$
(A5)	$\text{Woman}(\text{anna-magdalena})$	(A11)	$\text{Woman}(\text{maria-barbara})$
(A6)	$\text{marriedTo}(\text{johann-sebastian},$ $\text{anna-magdalena})$	(A12)	$\text{marriedTo}(\text{johann-ambrosius},$ $\text{maria-barbara})$

Table 3: Example OWL Ontology

(T1)	$\text{Person}(X)$:- $\text{Woman}(X)$.		$\text{hasChild}(X, f_1(X))$:- $\text{Father}(X)$.
(T2)	$\text{Person}(X)$:- $\text{Man}(X)$.		$\text{Person}(f_1(X))$:- $\text{Father}(X)$.
(T3)	$\text{Wife}(X)$:- $\text{Woman}(X)$,	(T6)	$\text{ancestorOf}(X,Y)$:- $\text{hasChild}(X,Y)$.
	$\text{marriedTo}(X,Y), \text{Husband}(Y)$.	(T7)	$\text{ancestorOf}(X,Y)$:- $\text{ancestorOf}(X,Y)$,
(T4)	$\text{Husband}(X)$:- $\text{Man}(X)$,		$\text{ancestorOf}(Y,Z)$.
	$\text{marriedTo}(X,Y), \text{Wife}(Y)$.	(T8)	$\text{marriedTo}(Y,X)$:- $\text{marriedTo}(X,Y)$.
(T5)	$\text{Father}(X)$:- $\text{Man}(X)$,	(T9)	$\text{LeipzigInhabitant}(X)$:- $\text{livesIn}(X, \text{leipzig})$.
	$\text{hasChild}(X,Y), \text{Person}(Y)$.	(T10)	$\text{Masterpiece}(Y)$:- $\text{Genius}(X)$,
	$\text{Man}(X)$:- $\text{Father}(X)$.		$\text{Composer}(X), \text{hasComposed}(X,Y)$.
(S1)	$=(f_1(X_1), f_1(X_2))$:- $=(X_1, X_2)$.	(S2)	$\text{Person}(X)$:- $\text{Person}(Y), =(X,Y)$.
...			
(A1)	$\text{I}_1(\text{johann-ambrosius})$.	(A2)	$\text{I}_2(\text{johann-sebastian})$.
	$\text{hasChild}(X, f_2(X))$:- $\text{I}_1(X)$.		$\text{Composer}(X)$:- $\text{I}_2(X)$.
	$\text{Man}(f_2(X))$:- $\text{I}_1(X)$.		$\text{Man}(X)$:- $\text{I}_2(X)$.
...			

Table 4: Translation of Example into LP

5 Practical Aspects

As our discussion of expressive relationships has made clear, there is a bi-directional semantic equivalence of (1) the DHL fragment of DL and (2) the DHL fragment of def-Horn. Likewise, there is a bi-directional semantic equivalence of the DLP fragment of DL and the DLP fragment of def-Horn. For our prime immediate goal is to enable rules (in LP / Horn) on top of ontologies (in DL) it is only required to provide one direction of *syntactic* mapping: from DL syntax to def-Horn syntax (and to the corresponding def-LP), rather than from def-Horn (or def-LP) to DL.

Besides enabling rules on top of DL-style ontologies it is desirable to exploit the relatively numerous, mature, efficient, scalable algorithms and implementations (i.e., engines) already available for LP inferencing so as to perform some fragment of DL inferencing. This immediately leads to the question which types of queries are typically supported by DL systems and how they can be supported by LP systems.

5.1 Inferencing

As discussed in the previous section, one of the prime goals of this work is to enable some fragment of DL inferencing to be performed by LP engines. In this section we will discuss the kinds of inference typically of interest in DL and LP, and how they can be represented in each other, i.e., in LP and DL respectively. Although the emphasis is on performing DL inferencing, via our mapping translation, using an LP reasoning engine, the reverse mapping can be used in order to perform LP inferencing using a DL reasoning engine. In particular, we will show how inferencing in (the DHL fragment of) DL can be reduced, via our translation, to inferencing in LP; and how vice versa, inferencing in (the DLP fragment of) LP can be reduced to inferencing in DL.

In a DL reasoning system, several different kinds of query are typically supported w.r.t. a knowledge base $\mathcal{K}$. These include queries about classes:

1. class-instance membership queries: given a class C,
   (a) ground: determine whether a given individual a is an instance of C;
   (b) open: determine all the individuals in $\mathcal{K}$ that are instances of C;
   (c) "all-classes": given an individual a, determine all the (named) classes in $\mathcal{K}$ that a is an instance of;
2. class subsumption queries: i.e., given classes C and D, determine if C is a subclass of D w.r.t. $\mathcal{K}$;
3. class hierarchy queries: i.e., given a class C return all/most-specific (named) superclasses of C in $\mathcal{K}$ and/or all/most-general (named) subclasses of C in $\mathcal{K}$;
4. class satisfiability queries, i.e., given a class C, determine if C is satisfiable (consistent) w.r.t. $\mathcal{K}$.

In addition, there are similar queries about properties: property-instance membership, property subsumption, property hierarchy, and property satisfiability. We will call $\mathcal{QDL}$ the language defined by the above kinds of DL queries.

In LP reasoning engines, there is one basic kind of query supported w.r.t. a ruleset $\mathcal{R}$: atom queries. These include:

1. ground: determine whether a ground atom A is entailed;
2. open (ground is actually a special case of this): determine, given an atom A (in which variables may appear), all the tuples of variable bindings (substitutions) for which the atom is entailed.

We call $\mathcal{QLP}$ the language defined by the above kinds of LP queries.

Next, we discuss how to reduce $\mathcal{QDL}$ querying in (the DHL fragment of) DL to $\mathcal{QLP}$ querying in (the DLP fragment of) LP using the mapping $\mathcal{T}$. We will assume that $\mathcal{R}$ is a ruleset derived from a DL knowledge base $\mathcal{K}$ via $\mathcal{T}$, and that all $\mathcal{QDL}$ queries are w.r.t. $\mathcal{K}$.

$\mathcal{QLP}$ (ground or open) atom queries can be used to answer $\mathcal{QDL}$ (ground or open) class-instance membership queries when the class is an $\mathcal{L}_h$-class, i.e., a is an instance of C iff $\mathcal{R}$ entails $\mathcal{T}(a:C)$. When C is an atomic class name, the mapping leads directly to a $\mathcal{QLP}$ atom query. When C is a conjunction, the result is a conjunction of $\mathcal{QLP}$ atom queries, i.e., a is an instance of $C \sqcap D$ iff $\mathcal{R}$ entails $\mathcal{T}(a:C)$ *and* $\mathcal{R}$ entails $\mathcal{T}(a:D)$. When C is a universal restriction, the mapping $\mathcal{T}(a:\forall P.C)$ gives $\mathcal{T}(C,y) \leftarrow P(a,y)$. This can be transformed into a $\mathcal{QLP}$ atom query using a simple kind of skolemisation, i.e., y is replaced with a constant b, where b is new in $\mathcal{R}$, and we have a is an instance of $\forall P.C$ iff $\mathcal{R} \cup \{P(a,b)\}$ entails $\mathcal{T}(b:C)$.

The case of property-instance membership queries is trivial as all properties are atomic: $\langle a,b \rangle$ is an instance of P iff $\mathcal{R}$ entails $P(a,b)$.

Complete information about class-instance relationships, to answer open or "all-classes" class-instance queries, can then be obtained via class-instance queries about all possible combinations of individuals and classes in $\mathcal{K}$. More efficient algorithms would no doubt be used in practise. For example, the set of named individuals and classes is known, and its size is worst-case linear in the size of the knowledge/rule base.

For $\mathcal{L}_h$-classes, $\mathcal{QDL}$ class subsumption queries can be reduced to $\mathcal{QLP}$ using a similar technique to class-instance membership queries, i.e., C is a subclass of D iff $\mathcal{R} \cup \{\mathcal{T}(a:C)\}$ entails $\mathcal{T}(a:D)$, for a new in $\mathcal{R}$. For $\mathcal{QDL}$ property subsumption queries, P is a subproperty of Q iff $\mathcal{R} \cup P(a,b)$ entails $Q(a,b)$, for a,b new in $\mathcal{R}$.

Complete information about the class hierarchy can be obtained by computing the partial ordering of classes in $\mathcal{K}$ based on the subsumption relationship.

In the DHL (and DLP) fragment, determining class/property satisfiability is a non-issue as, with the expressive power at our disposal in def-Horn, it is impossible to make a class or a property unsatisfiable.

Now let us consider the reverse direction from $\mathcal{QLP}$ to $\mathcal{QDL}$. In the DLP fragment of LP, every predicate is either unary or binary. Every atom query can thus be viewed as about either a named class or a property. Also, generally in LP, any open atom query

is formally reducible to a set of ground atom queries—one for each of its instantiations. Thus $\mathcal{QLP}$ is reducible to class-instance and property-instance membership queries in DL.

To recap, we have shown the following.

Theorem 5.1.1 (Inferencing Inter-operability) *For $\mathcal{L}_h$-classes, $\mathcal{QDL}$ querying in (the DHL fragment of) DL is reducible to $\mathcal{QLP}$ querying in (the DLP fragment of) LP, and vice versa.*

All of these queries can be reduced to one basic inference task: that of determining knowledge base satisfiability. An individual i is an instance of a class C iff adding $i : \neg C$ to $\mathcal{K}$ makes $\mathcal{K}$ unsatisfiable, and complete information about class-instance relationships could be obtained by applying this test to all possible combinations of individuals and classes in $\mathcal{K}$. Of course, more efficient algorithms would no doubt be used in practise. A class C is unsatisfiable iff adding $i : \neg C$ to $\mathcal{K}$ (for some individual i not already occurring in $\mathcal{K}$) makes $\mathcal{K}$ unsatisfiable, and C is a subclass of D iff $D \sqcup \neg C$ is unsatisfiable. Finally, complete information about the class hierarchy can be obtained by computing the partial ordering of classes in $\mathcal{K}$ based on the subsumption relationship.

The above queries can also be applied to properties, but are trickier to answer because DLs do not typically support property negation. Therefore, it is not possible to determine if a tuple $\langle i_1, i_2 \rangle$ is an instance of a property P by adding $\langle i_1, i_2 \rangle : \neg P$ to $\mathcal{K}$. The same effect can be achieved, however, by checking if adding $i : \exists P.\{j\}$ to $\mathcal{K}$ makes it unsatisfiable. Similarly, P is a subproperty of Q iff $\exists P.\{k\} \sqcap \neg \exists Q.\{k\}$ is unsatisfiable w.r.t. $\mathcal{K}$.

5.2 Inferencing with the DHL Extension

Here the translated logic programs may contain function symbols, denoting individuals whose existence is known, but whose name is not known.

Consider the ontology from Table 3. A query for instances of Man will return only johann-sebastian and johann-ambrosius. However, evaluating $\mathrm{Man}(X)$ in the program from the Table 4 will additionally return $f_2(\text{johann-ambrosius})$. This result set denotes the fact that in each model of $\mu[O]$ there is some unique element dependent on johann-ambrosius whose name is unknown. Note, however, that the presence of that element is very important for correctly inferring Father(johann-ambrosius). To obtain the same answer to a concept enumeration query as from a description logic reasoner, one must filter skolem constants out from the query result. In Prolog this can be implemented using the extralogical built-in predicate compound, which succeeds if the argument is bound to a complex term and not to a constant. Our query for the instances of Man can then be written as $\mathrm{Man}(X), \mathrm{not}(\mathrm{compound}(X))$. In rest of this paper we assume that this filtering is done externally to the logic program.

5.2.1 Recursive Definitions

The ontology from Table 3 contains recursive definitions (of concepts Husband and Wife). Generally, for such ontologies there are several possible interpretations. One typically considers only fixpoint interpretations – the interpretations where no new facts can be

inferred by applying the ontology definitions. Still, there are several different fixpoint interpretations of Husband and Wife: (1) as empty sets, (2) as {johann-ambrosius} and {maria-barbara} respectively, (3) as {johann-sebastian} and {anna-magdalena} respectively or (4) as union of (2) and (3). Now (1) is the least fixpoint (i.e. the fixpoint interpretation containing the smallest amount of facts), whereas (4) is the greatest fixpoint (i.e. the fixpoint interpretation containing the largest amount of facts) exist. In [23] properties of fixpoint interpretations have been analysed and another, so-called descriptive semantics, has been proposed as well. This semantics is important since for some expressive description logics fixpoints may not exist.

Similar problems arise when the DL ontology is translated into a logic program. Cycles in ontology definitions will manifest themselves as recursive rules, which can also be interpreted under least or greatest fixpoint semantics. However, most, if not all logic programming environments evaluate queries only under least fixpoint. It is possible, however, to evaluate logic programs without function symbols alternatively under the greatest fixpoint semantics. The authors of this paper are not aware of any known way to compute the descriptive semantics interpretation using logic programming. On the other hand, tableau procedures compute precisely this semantics [6]. To summarize, the approach presented in this paper is most suitable if the least fixpoint interpretation is sufficient for the requirements at hand.

5.3 Translation to relational databases

DLP programs can be implemented on top of relational databases. To perform this implementation all explicit facts of a predicate P are stored in a dedicated relation P_{Ext}. All non-recursive rules are translated to relational views. Rule bodies are translated to appropriate SQL queries (usually operating on other views). To obtain all explicit and implicit information, a view is defined to represent each predicate p. The query of the view integrates the explicit information, found in p_{ext} with the queries that represent the bodies of those rules where p is the head. The interested reader may refer to [29] for an in-depth description, algorithm and proof. Intuitively, this result follows from the following substitutions:

- each Datalog-rule can be simulated using the select-from-where construct of SQL;
- multiple rules defining the same predicate can be simulated using *union*; and
- negation in rule bodies can be simulated using *not in*.

To compute the answer for user queries the translated views are used. This realises a form of bottom up processing, since the queries involved in view definitions are performed on the extensional data and intermediate results are propagated up to a final query, which is the user query. This results in many irrelevant facts being computed in the intermediate steps; more efficient procedures based on sideways information passing have, however, been developed in the deductive database literature.

The above mentioned strategy is, however, not possible for recursively defined rules. Here additional processing is required.

5.4 Handling recursion

Modern relational database systems, which support the SQL:99 standard, can process some limited form of recursion, namely linear recursion with a path length one. Hence, the predicate used as the rule head may occur only once in the rule body. Cycles other than such linear self-references can also not be implemented.

Usually, binary recursive rules such as transitivity can be rewritten into a linear form. E.g. the mapping for transitive properties (see 1) can be rewritten into

$$P(x,y) \leftarrow P_{Ext}(x,y).$$
$$P(x,z) \leftarrow P_{Ext}(x,y) \wedge P(y,z).$$

The usual strategy to compute the remaining forms of recursive rules in relational databases is in-memory processing using some iterative strategy, e.g. the magic template procedure [24].

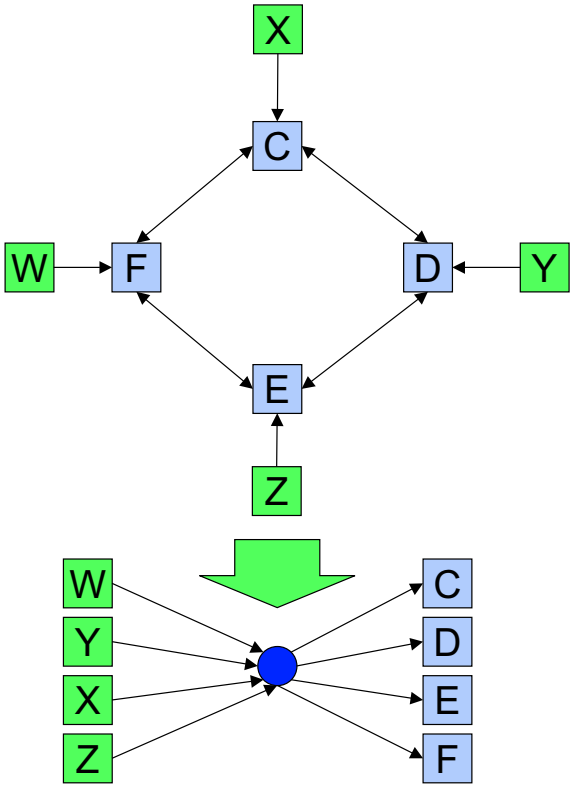

Figure 2: Cyclic Reference Removal

Indirect Recursion The remaining cases of non-linear recursion that cannot be rewritten into the SQL:99 constructs are mainly represented by the possibility of having cyclical class and property hierarchies.

We can, however, translate this case into the database by exploiting the observation that this form of recursion decomposes into unions, since no join processing of intermediate results (such as involved in computing the transitive closure of transitive properties)

is necessary. This is immediately clear for classes, since they are monadic predicates. A closer look at all axioms where binary predicates (properties) are in the head reveals the same. Hence, these cyclic references can be implemented via an algorithm that detects equivalence classes (each constituted by a cycle) in graphs. All incoming edges to an equivalence class must be duplicated to all members of the equivalence class, as illustrated in Figure 2. This may done by using a new intermediate predicate to collect the incoming edges and deriving the members of the equivalence class from this intermediate predicate. Afterwards, all rules that constitute the cyclic references within the equivalence class may safely be removed. The reader may note that this can also be performed (with appropriate adaptions) on the cyclic references imposed by inverse properties.

5.5 The KAON DLP prototype

We implemented a prototype system to demonstrate the feasability of our approach. The logic programming foundation is obtained by reusing the Datalog engine of KAON [5] – an ontology management infrastructure developed by FZI and AIFB at the University of Karlsruhe. The flow diagram showing how an OWL ontology is processed using the system is shown in figure 3.

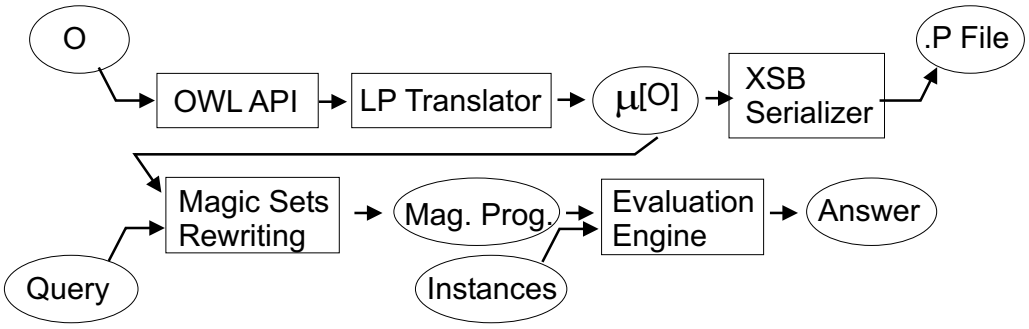

Figure 3: Prototype Block Diagram

The input is given by the means of the OWL API which is currently being developed within the WonderWeb project by the University of Manchester and the Institute AIFB. An ontology *O* is then passed to the Logic Program Translator which implementing the translation presented in Section 4, resulting in a logic program represented using the KAON datalog engine API.

This program can then be serialised in a number of formats, e.g. as a RuleML file. Alternatively, one can supply a query and instance data and then evaluate the program.

The implementation is intended to process larger quantities of data stored in relational databases, we have therefore opted for the bottom-up query evaluation technique, which matches well with the way how databases evaluate queries. However, bottom-up evaluation has the drawback that the entire program is evaluated, including the part not relevant to the query. In order to improve this, we apply the magic sets transformation [2]. Given a query, this transformation rewrites the program in a new program whose bottom-up evaluation will produce the same results with respect to the query. However, additional predicates introduced in the transformation ensure that only information relevant to the query is computed. In this respect the magic sets transformation simulates Prolog-style top-down computation through bottom-up computation.

6 Evaluation

In order to show the benefits of our approach, we have conducted a series of performance tests, the results of which we show in this section. We executed the tests with Racer version 1.7, XSB version 2.5 and our prototype described in the previous Section. We didn't use the FaCT system [15] since it doesn't support A-Box assertions.

About Tools. Racer [13] was chosen as the state-of-the-art description logic reasoning system employing the tableau decision procedure as the inference mechanism. The implementation language of Racer is LISP. XSB [26] is a well-known logic programming system using SLG-WAM resolution and tabling [28] as the inference mechanism. The implementation language of XSB is C. The implementation language of KAON and of our prototype is Java.

Test Assumptions. Many description logic systems use caching extensively in order to speed up query processing. For example, once Racer computes the extension of some concept, it caches the results, so the next time the same query is issued, it is answered almost immediately. We decided not to take this into account since we wanted to measure the performance of query answering alone. It is quite obvious that, if the answer to the query is cached, query answering will be fast. Moreover, Racer doesn't perform incremental maintenance of query answers. We have observed that whenever the A-Box is changed, even if the change doesn't affect the result of the query, Racer forgets all cached information and answering the query takes the same amount of time as the first time.

Answering queries using XSB also took much longer the first time since XSB had to compile the logic program. However, we decided to ignore this time. Compilation of the program is not the same as caching the query result – each time the program is executed, the query is evaluated from scratch. Further, in XSB it is possible to assert or retract facts programmatically and this doesn't influence the speed of query answering.

Test Procedure. Each test is characterised by a certain ontology structure and a concept whose extension is to be read. The ontology structure has been generated for different input parameters, resulting in ontologies of different sizes. Obtained ontologies have then been loaded in each of the tools and the query has been executed. The average of five such invocations has been taken as the performance measure for each test. If executing the query took more than 15 minutes, the test was interrupted – results of such tests are denoted with MAX in the Table 4.

Test Platform. We performed the tests on a standard PC with Pentium III processor running at 1.1 GHz, 380 MB of RAM running Windows XP operating system. Tests were written in Java and run using Sun's JDK version 1.4.1_01. Communication with Racer was done using JRacer library, whereas communication with XSB was performed through standard input and output.

Finally, before presenting the test results, we want to stress that we measured the performance of concrete tools. Although the algorithms used by all mentioned systems are

certainly important, the overall performance of the system is influenced by many other factors as well, such the quality of the implementation or the language used to implement the system. It is virtually impossible to exclude these factors from the performance measurement.

6.1 Measurement Results

First we give an overview of the types of tests we conducted. In describing tests we use *D* to denote the depth of the concept tree, *NS* to denote the number of subconcepts at each level in the tree, *NI* to denote the number of instances per concept and *P* to denote the number of properties. The results of tests 1 to 6 are presented in Figure 4, whereas the results of the test 7 are presented in Figure 5.

Test 1. The goal was to see how the very basic tasks of traversing the concept hierarchy are handled. The ontology structure was a symmetric tree of directly classified concepts. The test was performed for *D* = 3, 4, 5; *NS* = 5; *NI* = 10, 30; *P* = 0. The ontology contained no properties and the query involved computing the extension of one of the first level concepts.

Test 2. The goal of this test was to see how ontologies with larger number of properties are handled. The ontology structure from Test 1 was extended with one property per concept, which was instantiated for every third instance of the concept pointing to the next instance. However, the properties were not mentioned in concept definitions (i.e. they were not relevant to the query at all). The test was performed for *D* = 3, 4, 5; *NS* = 5; *NI* = 10 and the query again was to compute the extension of one of the first level concepts.

Test 3. In the previous test we observed that the performance of Racer depended on the number of property instances, even if these are not mentioned in concept definitions. Hence, in this test we wanted to see whether smaller number of properties, but larger number of property instances will make a difference. The ontology structure from Test 1 was extended with a fixed number of properties. Each instance was connected with the previously generated instance through one property. The test was performed for *D* = 3, 4, 5; *NS* = 5; *NI* = 10; *P* = 211.

Test 4. The goal of this test was to test answering a simple conjunctive query. The ontology structure was identical to the one from Test 3, but the query was $c1 \sqcap \exists p0.c12$. The test for performed for *D* = 3, 4; *NS* = 5; *NI* = 10; *P* = 3.

Test 5. The goal of this test was to see how simple concept definitions are handled. The ontology structure consisted of a fixed number of properties. Each concept in the concept tree was defined using the following axiom: $c_i \sqcup \exists p_k.c_{i-1} \sqsubseteq c$ (where c_i denotes i-th child of concept c).

Test 6. The goal of this test was to show how larger quantities of information can be efficiently managed by storing them in the database and applying the transformation presented in this paper. We repeated Test 1 for $D = 6$; $NS = 5$; $NI = 10$, but stored instances in the database. We used the ODBC bridge of XSB to access the data from the database and we implemented a JDBC interface for KAON. The test was executed only with XSB and KAON, since Racer doesn't have a database interface.

Test No.	Set	Concepts	Instances	Properties	Racer (s)	XSB (s)	KAON (s)
1	1	155	1550	0	12.18	**0.16**	0.18
	2	780	7800	0	95.17	**0.47**	0.83
	3	3905	39050	0	MAX	**2.11**	3.65
	4	155	4650	0	58.08	**0.36**	0.44
	5	780	23400	0	MAX	1.04	**0.98**
	6	3905	117150	0	MAX	MAX	**5.62**
2	1	155	1550	155	6.99	**0.15**	0.21
	2	780	7800	780	98.56	**0.43**	0.72
	3	3905	39050	3905	MAX	**2.27**	4.24
3	1	155	1550	211	5.39	**0.21**	0.39
	2	780	7800	211	MAX	**0.56**	0.88
	3	3905	39050	211	MAX	**2.34**	4.61
4	1	155	1550	3	34.30	**0.11**	0.82
	2	780	7800	3	MAX	**0.64**	3.07
5		155	1550	10	MAX	**0.49**	5.10
6		19530	195300	0	MAX	**14.60**	143.02

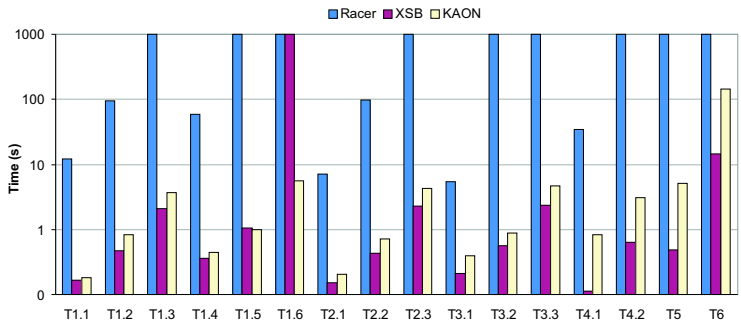

Figure 4: Results of Tests 1 to 6

Test 7. During execution of Test 3 we noticed that, although not relevant to the query result, the presence of property instances in the ontology has significant performance consequences in Racer. Hence, we repeated Test 3 with $D = 3$; $NS = 4$, while varying the number of instances per concept. We executed the test with ($P = 211$) and without properties ($P = 0$). We conducted the test with Racer only, since the goal was to show how presence of property instances significantly degrades the performance of reasoning in Racer. In XSB and KAON we observed no dependency of performance related to the number of property instances.

Test	Instances	No Prop. (s)	211 Prop. (s)
1	1550	10,48	4,63
2	3100	36,41	20,99
3	3225	48,94	30,24
4	3410	45,47	387,92
5	3875	71,06	624,48
6	4650	78,72	MAX

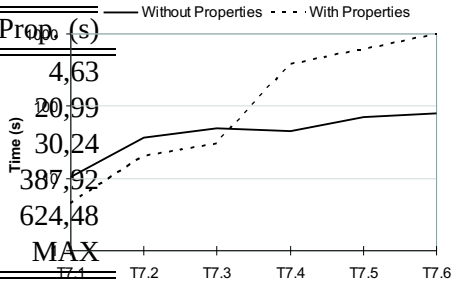

Figure 5: Results of Test 7

6.2 Discussion

From the results one can observe that the performance of Racer in all tests is significantly worse than the performance of XSB or KAON. We anticipate that this is mainly due to the fact that tableau reasoning procedure provides a proof or a refutation for one concept-instance pair, whereas SLG-WAM resolution or magic sets work with sets of instances. This is particularly true for answering conjunctive queries – SLG-WAM and magic sets evaluation take the full advantage of binding propagation, which limit the amount of computation to a minimum.

Further, one may observe that the performance of Racer deteriorates with the presence of property instances. For logic programs this makes absolutely no difference – the rules of the program don't reference predicates containing these property instances, so the amount of information stored in them isn't relevant. We don't have a plausible explanation for such behaviour in Racer.

Finally, one may observe that, as the amount of information grows, the information cannot be kept in main-memory, but must be stored in an external data storage. There we may observe that KAON still isn't as optimized as XSB. We presume that this stems from the fact that SLG-WAM resolution uses tabling, which prevents it from computing some answers twice. KAON, on the other hand, uses bottom-up computation of magic programs. In our case most predicates are unary, so binding passing can occur only in a limited way. Therefore, the magic sets rewriting worsens the performance, rather than improving it. Obviously, this is the point where KAON needs improvement.

7 Conclusion

We expect to help the Semantic Web to gain momentum by increasing both the size of the potential user base and the range of systems and applications able to exploit Semantic Web ontologies.

Besides the technical contribution, our translation will also help the logic programming user community to understand description logic principles, by relating them to principles familiar in logic programming. Moreover, the translated ontology can be a module in a larger logic program, for example allowing combination with other rules and thereby avoiding some of the limitations of reasoning in description logics. In particular, this allows the limitations imposed by the tree-like structure of description logic definitions to be overcome. The ability to interpret certain Description Logics-style ontologies through rules has numerous other benefits, amongst them are:

The Query language perspective Here, the capabilites of description logics with respect to instances is rather low. It cannot even express the least-expressive query language usually taken into account by database research - conjunctive queries [4]. This area is a strong hold of logic programming, which offers highly expressive constructs for instance reasoning. Hence, it is very promising to combine description logics with this paradigm to obtain the ability to state expressive instance queries on terminological knowledge bases.

The data integration perspective The majority of today's data resides in relational databases. This will not change when the Semantic Web grows. Most likely people will start exporting their data as RDF instances according to some ontology they have chosen. This essential leads to data that is replicated to enable ontology-based processing of that data. Today, the latter is done by reading some files into a classifier, such as FaCT [19] or Racer [13]. However, logic programming systems such as XSB [26] allow to access database data directly through built-in predicates. Furthermore, stratified Datalog programs, a restricted variant of logic programs with limited expressivity, can directly be implemented on top of SQL99-compliant relational databases. Hence, a LP-based implementation of OWL allows a closer interaction with live data.

The implementation perspective Currently, no full implementation of OWL is available. In order to become a success many implementations of OWL must be available. Many free and commercial implementations of logic programming systems are available. SQL99-compliant databases enjoy an even wider user community. This deliverable practically illustrates how these systems can be used as a basis for reasoning with OWL.

Motivated by these prospects, we presented an approach for translating a subset of OWL into logic programming. Based on this translation, we implemented a prototype system for handling OWL through logic programming. We conducted a series of tests comparing the performance of Racer, XSB and our own prototype system on ontologies of various sizes. From these tests we have observed that both XSB and our system perform on the order of magnitude better than Racer. This large difference in performance probably stems from the fact that description logics reasoners build a proof for one concept-instance pair at the time, whereas logic programming systems manage information in sets.

In future, we plan to investigate whether techniques of disjunctive deductive databases can be used to efficiently handle disjunction, negation and integrity constraints correctly. In particular, hyper-resolution and magic sets rewriting techniques seem promising in reducing query answering complexity in many practical cases.

References

[1] C. Baral and M. Gelfond, *Logic programming and knowledge representation*, Journal of Logic Programming **19/20** (1994), 73–148.

[2] C. Beeri and R. Ramakrishnan, *On the power of magic*, Proc. SIGMOD-SIGART Symposium on Principles of Database Systems, 1987, pp. 269–284.

[3] Tim Berners-Lee, *Cwm - close world machine*, Internet: http://www.w3.org/2000/10/swap/doc/cwm.html, 2002.

[4] Alexander Borgida, *On the relative expressiveness of description logics and predicate logics*, Artificial Intelligence **82** (1996), no. 1-2, 353–367.

[5] Erol Bozsak, Marc Ehrig, Siegfried Handschuh, Andreas Hotho, Alexander Maedche, Boris Motik, Daniel Oberle, Christoph Schmitz, Steffen Staab, Ljiljana Stojanovic, Nenad Stojanovic, Rudi Studer, Gerd Stumme, York Sure, Julien Tane, Raphael Volz, and Valentin Zacharias, *KAON - Towards a Large Scale Semantic Web*, E-Commerce and Web Technologies, Third International Conference, EC-Web 2002, Aix-en-Provence, France, September 2-6, 2002, Proceedings, 2002, pp. 304–313.

[6] M. Buchheit, F. M. Donini, and A. Schaerf, *Decidable Reasoning in Terminological Knowledge Representation Systems*, Journal of Artificial Intelligence Research **1** (1993), 109–138.

[7] D. Calvanese, G. de Giacomo, and M. Lenzerini, *On the Decidability of Query Containment under Constraints*, PODS-98, 1998, pp. 149–158.

[8] F. M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf, *Al-log: integrating datalog and description logics*, Journal of Intelligent Information Systems **10** (1998), 227–252.

[9] W. Dowling and J. Gallier, *Linear Time Algorithms for Testing the Satisfiability of Propositional Horn Formulae*, Journal of Logic Programming **3** (1984), 267–284.

[10] R. Fikes and D. McGuiness, *An axiomatic semantics for rdf, rdf schema and daml+oil*, Tech. Report KSL-01-01, KSL, Stanford University, 2001.

[11] E. Grädel, *On the Restraining Power of Guards*, JSL **64** (1999), 1719–1742.

[12] B. Grossof, I. Horrocks, R. Volz, and S. Decker, *Description Logic Programs: Combining Logic Programs with Description Logic*, Proceedings of WWW 2003 (Budapest, Hungary), May 2003.

[13] Volker Haarslev and Ralf Moller, *Description of the RACER system and its applications*, DL2001 Workshop on Description Logics, Stanford, CA, 2001.

[14] Steffen Hoelldobler, *Foundations of Equational Logic Programming*, LNAI, vol. 353, Springer, 1987.

[15] I. Horrocks, *The FaCT System*, Automated Reasoning with Analytic Tableaux and Related Methods: International Conference Tableaux'98, Springer-Verlag, pp. 307–312.

[16] I. Horrocks and U. Sattler, *Ontology Reasoning in the $\mathcal{SHOQ}(D)$ Description Logic*, IJCAI-01, 2001, pp. 199–204.

[17] I. Horrocks, U. Sattler, S. Tessaris, and S. Tobies, *How to decide Query Containment under Constraints using a Description Logic*, Proc. of LPAR'2000, 2000.

[18] I. Horrocks, U. Sattler, and S. Tobies, *Practical Reasoning for Expressive Description Logics*, Proc. 6th Int'l Conf. on Logic for Programming and Automated Reasoning (LPAR'99), no. 1705, Springer-Verlag, 1999, pp. 161–180.

[19] Ian Horrocks, Ulrike Sattler, and Stephan Tobies, *Practical reasoning for expressive description logics*, Proceedings of the 6th International Conference on Logic for Programming and Automated Reasoning (LPAR'99) (Harald Ganzinger, David McAllester, and Andrei Voronkov, eds.), no. 1705, Springer-Verlag, 1999, pp. 161–180.

[20] A. Y. Levy and M.-C. Rousset, *CARIN: A Representation Language Combining Horn Rules and Description Logics*, European Conf. on Artificial Intelligence, 1996, pp. 323–327.

[21] J. W. Lloyd, *Foundations of logic programming (second, extended edition)*, Springer series in symbolic computation, Springer-Verlag, New York, 1987.

[22] Mike Dean, Dan Connolly, Frank van Harmelen, James Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein, *Owl web ontology language 1.0 reference*, Internet: http://www.w3.org/TR/owl-ref/.

[23] B. Nebel, *Terminological Cycles: Semantics and Computational Properties*, Principles of Semantic Networks: Explorations in the Representation of Knowledge (J. F. Sowa, ed.), Morgan Kaufmann Publishers, San Mateo (CA), USA, 1991, pp. 331–361.

[24] R. Ramakrishnan, *Magic templates: A spellbinding approach to logic programs*, J. Logic Programming **11** (1991), 189–216.

[25] Jos De Roo, *Euler proof mechanism*, Internet: http://www.agfa.com/w3c/euler/, 2002.

[26] K. Sagonas, T. Swift, and D. S. Warren, *Xsb as an efficient deductive database engine*, Proc. of the 1994 ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'94) (R. T. Snodgrass and M. Winslett, eds.), 1994, pp. 442–453.

[27] K. Schild, *A Correspondence Theory for Terminological Logics: Preliminary Report*, IJCAI-91, 1991, pp. 466–471.

[28] Terrance Swift and David Scott Warren, *Analysis of SLG-WAM Evaluation of Definite Programs*, Symposium on Logic Programming, 1994, pp. 219–235.

[29] Jeffrey D. Ullman, *Principles of Database and Knowledge-base Systems*, vol. 1, Computer Science Press, 1988.

[30] M. Y. Vardi, *Why is Modal Logic So Robustly Decidable?*, Descriptive Complexity and Finite Models (N. Immerman and Ph. Kolaitis, eds.), American Mathematical Society, 1997.

[31] Youyong Zou, *Daml xsb interpretation*, Version 0.3, Internet: http://www.cs.umbc.edu/ yzou1/daml/damlxsb.P.txt, January 2001.