How to reason with OWL in a logic programming system

Markus Krötzsch, Pascal Hitzler, Denny Vrandečić
AIFB, Universität Karlsruhe, Germany

Michael Sintek
DFKI GmbH, Kaiserslautern, Germany

Abstract

Logic programming has always been a major ontology modeling paradigm, and is frequently being used in large research projects and industrial applications, e.g., by means of the F-Logic reasoning engine OntoBroker or the TRIPLE query, inference, and transformation language and system. At the same time, the Web Ontology Language OWL has been recommended by the W3C for modeling ontologies for the web. Naturally, it is desirable to investigate the interoperability between both paradigms. In this paper, we do so by studying an expressive fragement of OWL DL for which reasoning can be reduced to the evaluation of Horn logic programs. Building on the KAON*2 algorithms for transforming OWL DL into disjunctive Datalog, we give a detailed account of how and to what extent OWL DL can be employed in standard logic programming systems.* En route, *we derive a novel, simplified characterization of the supported fragment of OWL DL.*

1 Introduction

Logic programming has been a major ontology modeling paradigm since the advent of the Semantic Web. In particular F-Logic [11] as supported by different systems such as OntoBroker,[1] FLORA,[2] and TRIPLE[3] [19] were and are being used widely for ontology modeling in research and industry.

At the same time, the Web Ontology Language OWL [14] has been recommended by the W3C as standard for modeling complex ontologies on the web. Hence, there exist two competing major paradigms for expressive knowledge representation for the Semantic Web. This situation is also reflected by the Charter[4] of the W3C Rule Interchange Format Working Group formed in November 2005 as part of the W3C Semantic Web Activity: The charter refers explicitly to several W3C member submissions, including the Semantic Web Rule Language SWRL [8] and the Web Rule Language WRL [1]. While the former adheres to the conceptual design decisions underlying OWL DL, the latter is based on F-Logic.

The two paradigms differ in many respects. While OWL DL is decidable, F-Logic is not.[5] While F-Logic is a Turing-complete programming paradigm, OWL DL can mainly be used for specifying static knowledge bases. Probably the most prominent conceptual difference is that OWL DL adheres to the open world assumption, while F-Logic – and logic programming in general – is committed to the closed world assumption. Indeed, OWL DL is based on description logics and is thus basically a decidable fragment of first-order predicate logic, from which it borrows its formal semantics. The semantics of logic programming based languages like F-Logic differs from that of first-order logic in certain ways, and this often also results in a procedural flavor. OWL and standard logic programming paradigms [13] thus differ in modeling styles and formal semantics in ways which go beyond syntactic issues. The choice of modeling language indeed inflicts heavily on the treatment of implicit knowledge by means of reasoning support [7].

While – in an ideal universe – a single ontology representation language might be a desirable goal, it is only realistic to expect that the currently evolving Semantic Web will be highly heterogeneous, not only in terms of the knowledge represented, but also in terms of ontology language paradigms used. It is desirable, therefore, to embark on the creation of integrated reasoning tools which can cope with knowledge represented in different paradigms and thus establish language interoperability in a practical sense. Accordingly, various approaches have been proposed for integrating logic programs and description logics [3, 2, 5, 17].

The study of language interoperability is a delicate issue. Ideally, a general logic would be established within which different ontology languages could be embedded semantically. To date, however, it is rather unclear whether a satisfactory such logic can and will be found, and whether it will be possible to put such a logic to practical use.

[1] `http://ontobroker.semanticweb.org`
[2] `http://flora.sourceforge.net`
[3] `http://triple.semanticweb.org`
[4] `http://www.w3.org/2005/rules/wg/charter`

[5] Indeed, logic programming with non-monotonic negation is usually not even semi-decidable. See [18] for an account of the well-founded semantics.

So instead of pursuing the search for a common generalization of different paradigms, we rather embark on the complementary quest for large fragments which can be mapped semantically into other languages, in the sense that they can be dealt with procedurally by corresponding tools. More precisely, we ask to what extent ontologies from one paradigm can be processed correctly by an engine from another paradigm. In this paper, we investigate this for OWL and logic-programming based engines, such as XSB Prolog,[6] which can be understood as the basic underlying paradigm for, e.g., OntoBroker, FLORA, and TRIPLE.

Much of the content of this paper is implicitly contained in the Ph.D. thesis [15]. However, it takes considerable effort to trim the rather involved theory down to such an extent that the results become accessible, and so we chose to share our insights. The original contributions of the paper are as follows.

- We provide a novel characterization for the OWL language fragment which can be translated into logic programs.
- We give a crisp and accessible account of how to realize the interoperability.
- We provide a specialized conversion tool for using the approach.

The paper is structured as follows. In Section 2 we present an overview of our work within some historic context. In Section 3 we provide the novel characterization for the mentioned OWL language fragment Horn-$\mathcal{SHIQ}$, which we can translate into logic programs, and discuss some issues arising in this context. In Section 4 we discuss our tool which realizes the approach, and we provide a comprehensive example that illustrates the translation. Language interoperability within our approach is discussed in Section 5. We close with conclusions in Section 6.

Acknowledgments. The authors wish to thank Boris Motik for the numerous clarifying discussions during the writing of this paper. He would deserve to be a co-author, but chose not to. Furthermore, support by the German Federal Ministry of Education and Research (BMBF) under the SmartWeb project (grant 01 IMD01 B), by the Deutsche Forschungsgemeinschaft (DFG) under the ReaSem project, and by the European Commission under contract IST-2003-506826 SEKT is gratefully acknowledged.

2 OWL DLP, and how to do better

The straightforward candidate for an OWL DL fragment to be processed by a logic programming system is DLP (Description Logic Programs), as introduced in [4] and studied in detail in [20]. DLP is a naive Horn fragment of OWL DL, and although it is of very limited expressiveness compared with OWL DL, it appears to be a suitable core paradigm for some practical purposes [6].

As sets of Horn clauses, DLP ontologies can be expressed as logic programs without negation, and although their OWL semantics differs somewhat from their reading under logic programming semantics [13], these two perspectives are compatible up to a certain extent [5]. From an interoperability perspective, however, DLP is of limited use as it is only a small fragment of OWL DL.

However, a larger OWL fragment than OWL DLP can be mapped into clausal form by means of sophisticated algorithms developed for the KAON*2* OWL reasoner.[7] The result of this transformation is disjunctive Datalog without function symbols, and this is done in such a way that the resulting disjunctive Datalog programs can be processed in a standard Datalog fashion without loosing any of the OWL DL semantics of the original knowledge base. In order to improve on OWL DLP, and in order to find a larger fragment which is processable by a logic programming system, a natural starting point is thus the OWL DL fragment which is translated to Horn clauses by means of the KAON*2* translation algorithms. This fragment was called Horn-$\mathcal{SHIQ}$ in [10]. In the following, we will basically analyze the Horn-$\mathcal{SHIQ}$ fragment regarding the question as to what extent it can be dealt with within a standard logic programming system like XSB Prolog.

3 From OWL to Logic Programs

We next present and discuss the transformation of OWL DL specifications into Horn logic programs as provided by the KAON*2* algorithm described in [9, 15], which compiles $\mathcal{SHIQ}$ knowledge bases into (function- and negation-free) *disjunctive Datalog*.[8] The usage of this algorithm for our purpose of casting OWL DL into Horn clauses is restricted in two ways. On the one hand, the algorithm was devised to handle the description logic $\mathcal{SHIQ}$(D),[9] while OWL DL corresponds to $\mathcal{SHOIN}$(D). This means that nominals are not supported yet, and thus are excluded from our treatment. On the other hand, a disjunctive Datalog program that is obtained from the transformation is in general not processable by logic programming system for Horn programs.

[6] `http://xsb.sourceforge.net`

[7] `http://kaon2.semanticweb.org`

[8] A (function- and negation-free) disjunctive Datalog rule is a function-symbol-free formula of the form $A_1 \vee \ldots \vee A_n \leftarrow B_1 \wedge \ldots \wedge B_m$. This in turn is equivalent to the formula $A_1 \vee \ldots \vee A_n \vee \neg B_1 \wedge \ldots \wedge \neg B_m$ in conjunctive (or *clausal*) normal form. Any first order predicat logic formula that does not contain quantifiers and functions symbols can be cast in such a Datalog rule by means of standard algorithms.

[9] At the time of the writing of this paper, the conceived support for concrete domains is not implemented yet, i.e., the program handles $\mathcal{SHIQ}$ instead of $\mathcal{SHIQ}$(D).

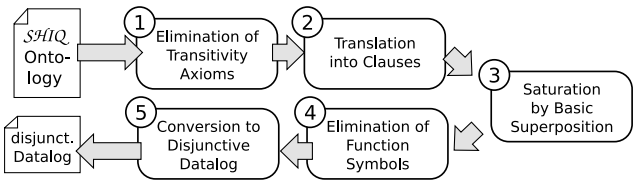

Figure 1. Algorithm for reducing $\mathcal{SHIQ}$ to disjunctive Datalog.

The latter problem of incompatibility with Horn logic reasoners can have various causes, each of which will be discussed individually below. First of all, one obviously has to ensure that none of the created rules have a disjunction in the head, i.e., that each clause contains at most one non-negated literal. The $\mathcal{SHIQ}$ fragment for which this is the case still is surprisingly large and has been discussed in [10]. In Section 3.2 below, we review these results and derive a substantially simpler alternative characterization. While this restriction eliminates disjunctions from the rule heads, it still leaves the possibility of having rules with empty heads, also known as *integrity constraints*. Section 3.3 discusses how to model such constraints in a (strict) Horn-logic setting. Another problem is the *equality* predicate encountered in many OWL DL specifications, since logic programming systems do usually not provide built-in equality. In Section 3.4, we show how this can be taken into account by the translation. Finally, the sophisticated algorithms of KAON*2* enable us to deal with some OWL DL specifications that are not in the fragment of DLP. In particular, the system supports arbitrary existential quantifiers, as is discussed in the next section.

3.1 The KAON*2* Transformation Algorithm

Here, we sketch how the transformation algorithm of KAON*2* works in principle. An exhaustive treatment can be found in [15]. The general workflow for transforming a $\mathcal{SHIQ}$ knowledge base into a disjunctive Datalog program is depicted in Fig. 1. The steps of the algorithm can roughly be described as follows. (1) Transitivity axioms for roles S are replaced by adding axioms of the form $\forall R.C \sqsubseteq \forall S.\forall S.C$ whenever $S \sqsubseteq R$. This is a standard method for eliminating transitivity axioms, and the resulting knowledge base is satisfiable if and only if the original knowledge base is.

Employing the fact that $\mathcal{SHIQ}$ can be regarded as a subset of first-order logic, step (2) uses standard algorithms to transform the knowledge base into conjunctive normal form. This involves eliminating existential quantifiers by *Skolemization*, and thus function symbols must be introduced into the knowledge base.

Next, in step (3), the obtained set of clauses is partially *saturated* by adding logical consequences. This is the crucial step of the algorithm where one has to compute enough consequences to allow for a reduction to function-free Datalog. Since the computational complexity is ExpTime for $\mathcal{SHIQ}$ but only NP for disjunctive Datalog, it should not come as a surprise that this transformation step can be exponential in the size of the input. The details of this step are rather sophisticated and we refer interested readers to [15] for details and proofs.

Now function symbols can safely be eliminated in step (4). To ensure that this process still preserves satisfiability of the knowledge base, one has to add a linear number of auxiliary axioms. Finally, it is easy to syntactically transform the resulting set of clauses into a disjunctive Datalog program in step (5).

Due to the transformations in steps (1) and (2), the output of the algorithm is in general not semantically equivalent to the input. Since all major reasoning tasks for $\mathcal{SHIQ}$ can be reduced to satisfiability checking, it is sufficient for the transformation to preserve satisfiability. However, combining the Datalog program with further axioms of first-order logic generally destroys completeness of reasoning, i.e., the resulting knowledge base might be (first-order) satisfiable under some model that is not a model for the original OWL ontology. In combination with a nonmonotonic reasoning paradigm, such incompleteness often leads to unsoundness as well. Fortunately, there are a number of axioms that one can safely add without jeopardizing soundness or completeness, as discussed in Section 5 below. Yet, in general, completeness of reasoning in the presence of additional axioms can only be restored by applying the transformation algorithm to the whole knowledge base again.

3.2 Horn-$\mathcal{SHIQ}$

Now, we turn to the question which knowledge bases are transformed exclusively into rules that do not contain disjunctions in the head. This class of knowledge bases has already been introduced as Horn-$\mathcal{SHIQ}$ in [10], but in the following we derive a novel characterization which greatly simplifies the presentation. Generally, the description of Horn-$\mathcal{SHIQ}$ exploits the insight that, whenever the normal form transformation yields only Horn clauses, any relevant clause that is obtained by the saturation step (3) is also Horn. Thus, it suffices to identify those $\mathcal{SHIQ}$-axioms that produce Horn-clauses after step (2) of the transformation algorithm.

When defining Horn-$\mathcal{SHIQ}$, one needs to consider the details of the normal form transformation implemented in KAON*2*. In particular, KAON*2* performs a structural transformation that introduces new names for subformulae. For example, the axiom $A \sqsubseteq \exists R.\exists R.C$ is transformed into ax-

$C\|_\epsilon$	$= C$	$\mathsf{pol}(C, \epsilon)$	$= 1$
$(\neg C)\|_{1p}$	$= C\|_p$	$\mathsf{pol}(\neg C, 1p)$	$= -\mathsf{pol}(C, p)$
$(C_1 \circ C_2)\|_{ip}$	$= C_i\|_p$	$\mathsf{pol}(C_1 \circ C_2, ip)$	$= \mathsf{pol}(C_i, p)$
		for $\circ \in \{\sqcap, \sqcup\}$ and $i \in \{1, 2\}$	
$\Diamond R.C\|_{2p}$	$= C\|_p$	$\mathsf{pol}(\Diamond R.C, 2p)$	$= \mathsf{pol}(C, p)$
		for $\Diamond \in \{\forall, \exists\}$	
$\leq n\, R.C\|_{3p}$	$= C\|_p$	$\mathsf{pol}(\leq n\, R.C, 3p)$	$= -\mathsf{pol}(C, p)$
$\geq n\, R.C\|_{3p}$	$= C\|_p$	$\mathsf{pol}(\geq n\, R.C, 3p)$	$= \mathsf{pol}(C, p)$

Table 1. Positions in a concept (left) and their polarity (right).

D	$\mathsf{pl}^+(D)$	$\mathsf{pl}^-(D)$
$\bot$	0	0
$\top$	0	0
A	1	0
$\neg C$	$\mathsf{pl}^-(C)$	$\mathsf{pl}^+(C)$
$\bigsqcap C_i$	$\max_i \mathsf{sgn}(\mathsf{pl}^+(C_i))$	$\sum_i \mathsf{sgn}(\mathsf{pl}^-(C_i))$
$\bigsqcup C_i$	$\sum_i \mathsf{sgn}(\mathsf{pl}^+(C_i))$	$\max_i \mathsf{sgn}(\mathsf{pl}^-(C_i))$
$\exists R.C$	1	$\mathsf{sgn}(\mathsf{pl}^-(C))$
$\forall R.C$	$\mathsf{sgn}(\mathsf{pl}^+(C))$	1
$\geq n\, R.C$	1	$\frac{n(n-1)}{2} + n\ \mathsf{sgn}(\mathsf{pl}^-(C))$
$\leq n\, R.C$	$\frac{n(n+1)}{2} + (n+1)\ \mathsf{sgn}(\mathsf{pl}^-(C))$	1

Table 2. Definition of $\mathsf{pl}^+(D)$ and $\mathsf{pl}^-(D)$.

$$\mathbf{C_0^+} ::= \top \mid \bot \mid \neg \mathbf{C_0^-} \mid \mathbf{C_0^+} \sqcap \mathbf{C_0^+} \mid \mathbf{C_0^+} \sqcup \mathbf{C_0^+} \mid \forall \mathbf{R}.\mathbf{C_0^+}$$

$$\mathbf{C_0^-} ::= \top \mid \bot \mid \neg \mathbf{C_0^+} \mid \mathbf{C_0^-} \sqcap \mathbf{C_0^-} \mid \mathbf{C_0^-} \sqcup \mathbf{C_0^-} \mid \exists \mathbf{R}.\mathbf{C_0^-} \mid \mathbf{A}$$

$$\mathbf{C_1^+} ::= \top \mid \bot \mid \neg \mathbf{C_1^-} \mid \mathbf{C_1^+} \sqcap \mathbf{C_1^+} \mid \mathbf{C_0^+} \sqcup \mathbf{C_1^+} \mid \exists \mathbf{R}.\mathbf{C_1^+} \mid \forall \mathbf{R}.\mathbf{C_1^+} \mid \geq n\, \mathbf{R}.\mathbf{C_1^+} \mid \leq 1\, \mathbf{R}.\mathbf{C_0^-} \mid \mathbf{A}$$

$$\mathbf{C_1^-} ::= \top \mid \bot \mid \neg \mathbf{C_1^+} \mid \mathbf{C_0^-} \sqcap \mathbf{C_1^-} \mid \mathbf{C_1^-} \sqcup \mathbf{C_1^-} \mid \exists \mathbf{R}.\mathbf{C_1^-} \mid \forall \mathbf{R}.\mathbf{C_1^-} \mid \geq 2\, \mathbf{R}.\mathbf{C_0^-} \mid \leq n\, \mathbf{R}.\mathbf{C_1^+} \mid \mathbf{A}$$

Table 3. A grammar for defining Horn-$\mathcal{ALCHIQ}$. A and R denote the sets of all concept names and role names, respecitvely.

ioms $A \sqsubseteq \exists R.Q$ and $Q \sqsubseteq \exists R.C$, where Q is a new concept name. When done with care, such translations can help to retain Hornness of a large class of clauses. Further details on structural transformation (which, in general, does only preserve satisfiability) are found in [15].

We recall the definition of Horn-$\mathcal{SHIQ}$ as given in [10], which requires us to introduce a couple of auxiliary concepts first. Subconcepts of some description logic concept are denoted by specifying their *position*. Formally, a position p is a finite sequence of integers, where ϵ denotes the empty position. Given a concept D, $D|_p$ denotes the *subconcept* of D at position p, defined recursively as in Table 1 (left). In this paper, we consider only positions that are defined according to this table, and the set of *all positions in a concept* D is understood accordingly. Given a concept D and a position p in D, the *polarity* $\mathsf{pol}(D, p)$ of D at position p is defined as in Table 1 (right). Using this notation, we can state the following definition of Horn knowledge bases.

Defintion 1 ([10, Definition 1]) *Let* pl^+ *and* pl^- *denote mutually recursive functions that map a* $\mathcal{SHIQ}$ *concept* D *to a non-negative integer as specified in Table 2, where* $\mathsf{sgn}(0) = 0$ *and* $\mathsf{sgn}(n) = 1$ *for* $n > 0$*. We define a function* pl *that assigns to each* $\mathcal{SHIQ}$*-concept* D *and position* p *in* D *a non-negative integer by setting:*

$$\mathsf{pl}(D, p) = \begin{cases} \mathsf{pl}^+(D|_p) & \text{if } \mathsf{pol}(D, p) = 1, \\ \mathsf{pl}^-(D|_p) & \text{if } \mathsf{pol}(D, p) = -1, \end{cases}$$

A concept C *is* Horn *if* $\mathsf{pl}(C, p) \leq 1$ *for every position* p *in* C *(including the empty position* ϵ*). An extensionally reduced*[10] $\mathcal{ALCHIQ}$ *knowledge base* KB *is* Horn *if* $\neg C \sqcup D$ *is Horn for each axiom* $C \sqsubseteq D$ *of* KB*.*

While suitable as a criterium for *checking* the Hornness of single axioms or knowledge bases, this definition is not particularly suggestive as a description of the class of Horn knowledge bases as a whole. Indeed, it is not readily seen for which formulae pl yields values smaller or equal to 1 for all possible positions in the formula. On the other hand, Definition 1 is still overly detailed as pl calculates the *exact* number of positive literals being introduced when transforming some (sub)formula.

In order to derive a more convenient characterization, observe that, since we require the value of pl to be smaller or equal to 1 at *all* positions of the concept, there cannot be any sub-concepts of higher values, even though the value of the subconcept is not decisive for some cases of the calculation of pl (e.g. for $\mathsf{pl}^+(\exists R.C)$). Thus we can generally restrict to concepts with a pl-value ≤ 1. To do so, one has to consider four different classes of concepts, having a pl-value either $=0$ or ≤ 1 when occuring either at a positive or negative position in a formula. Appropriate classes of $\mathcal{ALCHIQ}$ concepts are defined in Table 3, where notation is simplified by omitting concepts that are obviously equivalent to those already included. For example, we use $\exists R.C$ instead of $\geq 1\, R.C$, and exploit commutativity and associativity of $\sqcap$ and $\sqcup$.

Intuitively, the classes $\mathbf{C_0^+}$ and $\mathbf{C_1^+}$ define exactly those concepts for which the value of pl is smaller or equal to 0 and 1, respectively. In particular, $\mathbf{C_1^+}$ denotes the class of all Horn concepts. Let us now show this formally.

Lemma 1 *A* $\mathcal{SHIQ}$ *concept* D *is in* $\mathbf{C_0^+}$ *(*$\mathbf{C_0^-}$*) iff we find, for every position* p *in* D *(in* $\neg D$*), that* $\mathsf{pl}(D, p) = 0$ *(*$\mathsf{pl}(\neg D, p) = 0$*).*

Proof: Observe that using $\neg D$ in the condition for $\mathbf{C_0^-}$ re-

[10] A knowledge base is *existentionally reduced* if its ABox contains only literal concepts. This can always be achieved by introducing new names for complex concept terms.

flects the fact that those concepts occur only at negative subpositions in concepts of type $\mathbf{C}_\mathbf{0}^+$. The proof proceeds by induction over the structure of concepts. For the base cases $\bot$, $\top$, and $\mathbf{A}$, the claim is obvious. Now let $D = \neg C$. It is easy to see that $D \in \mathbf{C}_\mathbf{0}^+$ iff $C \in \mathbf{C}_\mathbf{0}^-$. By the indcution hypothesis, this is equivalent to $\mathsf{pl}(D, p) = \mathsf{pl}(\neg C, p) = 0$ for any p. Conversely, $D \in \mathbf{C}_\mathbf{0}^-$ iff $C \in \mathbf{C}_\mathbf{0}^+$, which is equivalent to $\mathsf{pl}(C, p) = 0$ for every p in C. By the definition of pl^-, it is easy to see that this is equivalent to $\mathsf{pl}(\neg D, p) = 0$ for every p in $\neg D$.

The remaining cases are very similar, and the arguments for $\mathbf{C}_\mathbf{0}^+$ and $\mathbf{C}_\mathbf{0}^-$ are mostly symmetric. We exemplify the required reasoning by considering the case $D = \exists R.C$. Clearly, $\mathsf{pl}(D, \epsilon) \neq 0$, so D is not in $\mathbf{C}_\mathbf{0}^+$. On the other hand, we find that $D \in \mathbf{C}_\mathbf{0}^-$ iff $C \in \mathbf{C}_\mathbf{0}^-$ iff $\mathsf{pl}(\neg C, p) = 0$ for every p. By the definition of pl^-, this clearly is equivalent to $\mathsf{pl}(\neg D, p) = 0$ for every p. All other cases are shown analogously. □

The following is the crucial result for our characterization.

Proposition 1 *A $\mathcal{SHIQ}$ concept D is in $\mathbf{C}_\mathbf{1}^+$ ($\mathbf{C}_\mathbf{1}^-$) iff we find, for every position p in D (in $\neg D$), that $\mathsf{pl}(D, p) \leq 1$ ($\mathsf{pl}(\neg D, p) \leq 1$).*

Proof: The proof proceeds as in Lemma 1, so we only consider some specific cases of the induction. So assume that $D = C_1 \sqcup C_2$. Then $D \in \mathbf{C}_\mathbf{1}^+$ iff $C_1 \in \mathbf{C}_\mathbf{0}^+$ and $C_2 \in \mathbf{C}_\mathbf{1}^+$. By the induction hypothesis and the definition of pl^+, we obtain $\mathsf{pl}(D, p) \leq 1$ for all p in D.

Conversely, assume that $\mathsf{pl}(D, p) \leq 1$ for all p in D. For this to hold at all positions other than ϵ, C_1 and C_2 must be in $\mathbf{C}_\mathbf{1}^+$. In addition, $\mathsf{pl}(D, \epsilon) = \mathsf{pl}^+(C_1) + \mathsf{pl}^+(C_2) \leq 1$ implies that $\mathsf{pl}^+(C_1) = 0$ or $\mathsf{pl}^+(C_2) = 0$. Without loss of generality, we assume that $\mathsf{pl}^+(C_1) = 0$. By the definition of pl^+ and pl^-, it is easy to see that this implies $\mathsf{pl}(C_1, p) = 0$ for all p in C_1. Thus, by Lemma 1, $C_1 \in \mathbf{C}_\mathbf{0}^+$. The case for $D = C_1 \sqcup C_2$ and $\mathbf{C}_\mathbf{1}^-$ is simpler, since values of C_1 and C_2 are combined with the max operation here. All other cases are shown analogously.□

Now we can sum up our results in the following corollary.

Corollary 1 *An extensionally reduced $\mathcal{ALCHIQ}$ knowledge base KB is Horn iff, for all axioms $C \sqsubseteq D$ of KB, one finds that $(\neg C \sqcup D) \in \mathbf{C}_\mathbf{1}^+$.*

We argue that this definition is more easy to comprehend than the original characterization. For example, it is now readily seen that the axiom $\geq 2\, R.(C \sqcap \exists R.D) \sqsubseteq \forall R.\neg E$ is of the form $\mathbf{C}_\mathbf{1}^- \sqsubseteq \mathbf{C}_\mathbf{0}^+$ and is thus Horn, whereas $\forall R.\neg E \sqsubseteq \geq 2\, R.(C \sqcap \exists R.D)$ is not. This is less obvious when considering the original definition. Also note that Corollary 1 only depends on Table 3, but does not require the definition of *position*, *polarity*, or any of the auxiliray functions $\mathsf{pl}^{(+/-)}$. It is possible to further extend the above characterisation to $\mathcal{SHIQ}$, and this has been done in [12].

So far, we only considered $\mathcal{ALCHIQ}$ knowledge bases, i.e. we excluded transitivity from our treatment. The reason is that transitivity axioms of $\mathcal{SHIQ}$ are replaced by $\mathcal{ALCHIQ}$ axioms before axioms are transformed into clausal normal form. These additional axioms can actually lead to non-Hornness as well. We refrain form giving a more precise description, which is readily obtained by combining our results for $\mathcal{ALCHIQ}$ with the processing of transitivity axioms described in [15].

3.3 Integrity Constraints

Even when restricting to Horn-$\mathcal{SHIQ}$, the transformation algorithm can produce rules of the form $\leftarrow B_1 \wedge \ldots \wedge B_n$ that do not have a head at all. These rules correspond to clauses of the form $\neg B_1 \vee \ldots \vee \neg B_n$, and thus can be regarded as *integrity constraints* asserting that the statements B_i can never become true simultaneously. A typical example are the disjointness-conditions of classes in OWL DL, e.g., the statement $C \sqcap D \equiv \bot$ translates to $\leftarrow C(x) \wedge D(x)$.

While logic programming systems often do not support such rules,[11] this does not impose real restrictions on our translation. To see why this is the case, recall that the logical consequences of a Horn-program are obtained by considering its least (Herbrand) model: an atomic statement is a consequence of a Horn-program if and only if it is entailed by the least model. On the other hand, the role of integrity constraints is to disallow certain interpretations for the knowledge base. In the case of Horn-logic this means that given integrity constraints either eliminate the least model, thus making the whole theory inconsistent, or otherwise have no effect on the reasoning at all.

Therefore, it is not necessary to extend logic programming systems with special capabilities to support integrity constraints. Instead, we just translate clauses $\neg B_1 \vee \ldots \vee \neg B_n$ into rules of the form $\mathsf{inc} \leftarrow B_1 \wedge \ldots \wedge B_n$, where inc is a freshly introduced nullary "inconsistency" predicate. One can now query the system for inc: if the answer is "yes" then inc is true in the least model, and thus every model necessarily violates one of the constraints. If inc is not entailed, then the knowledge base is consistent and can be queried as usual.

3.4 Equality

Equality plays an important role in many description logics. The reason is that description logics have a classical first-order semantics without a unique name assumption.

[11] Rules with an empty head typically only appear as queries.

$$X \approx X, \quad X \approx Y \leftarrow Y \approx X, \quad X \approx Z \leftarrow X \approx Y \wedge Y \approx Z$$
$$C(Y) \leftarrow C(X) \wedge X \approx Y \quad \text{for every concept name } C$$
$$R(Y_1, Y_2) \leftarrow R(X_1, X_2) \wedge X_1 \approx Y_1 \wedge X_2 \approx Y_2 \quad \text{for every role name } R$$

Table 4. Horn axioms for describing equality.

Thus it is possible that syntactically different logical constants denote the same element in a model. This can be stated explicitly in the form of ABox statements such as "$a \approx b$" but it can also be concluded indirectly during reasoning.

The latter occurs when number restrictions appear in the knowledge base: given the axioms $\leq 1R(a)$, $R(a, b)$, and $R(a, c)$, we conclude that b and c must denote the same element. Indeed, it is standard to treat number restrictions by translating them into logical formulae that include equality statements. For example, the statement $\leq nR(a)$ translates to $\forall x_1, \ldots, x_{n+1}.R(a, x_1) \wedge \ldots \wedge R(a, x_{n+1}) \rightarrow x_1 \approx x_2 \vee x_1 \approx x_3 \vee \ldots \vee x_n \approx x_{n+1}$, where the consequent denotes the disjunction of all pairwise equality statements among the variables. This is also the way how KAON*2* treats number restrictions and thus these are closely tied to the support for equality.

Unfortunately, equality reasoning in the presence of functions symbols is a problem of formidable difficulty, and an almost inevitable source of non-termination. Thus common logic programming systems typically do not provide a native support for a general equality predicate. However, it is well known that one can axiomatically describe an equality predicate in Horn logic, as long as only finitely many predicate symbols are considered. The corresponding clauses are depicted in Table 4. In order to ensure that these rules do not impair decidability, one must slightly extend them to become *DL-safe*, as explained in Section 5. In general, this restriction is also required for the proofs of completeness of reasoning. Here, we only remark that with the (DL-safe version of the) above rules, equality reasoning is possible within logic programming systems.

4 Implementation and example

A precompiled binary distribution of the KAON*2* implementation is available online, and can be downloaded free of charge for research and academic purposes.[12] KAON*2* is a whole infrastructure for managing and reasoning with ontologies, accessible via an API rather than through some fixed user interface. It includes methods for obtaining the basic disjunctive Datalog program, but does not yet incorporate the specific adjustments to logic programming systems described in the previous section. Therefore, we provide an additional wrapper application *dlpconvert* [16] as part of the KAON*2 OWL Tools*.[13] The latter is a collection of command line tools applicable for tasks ranging from satisfiability testing to the conversion of OWL specifications into LaTeX-processable logical formulae. In the following, we briefly discuss the usage of *dlpconvert* and demonstrate the algorithm for a concrete example.

In general, *dlpconvert* reads an ontology in OWL/RDF or OWL/XML format and transforms it into a logic program in Prolog syntax. If invoked without further arguments, this transformation does not support Skolemization of existential quantifiers or any form of non-Horn clauses (thus defining a form of "DLP" based on Horn-$\mathcal{SHIQ}$). If the `-X` switch is used, *dlpconvert* uses the full capabilities of KAON*2* to produce disjunctive Datalog from arbitrary $\mathcal{SHIQ}$ input.

There is also an `-flogic` option that syntactically adjusts the output for use in an F-Logic system. In particular, this involves the usage of F-Logic's "object oriented" syntactic features, such as $C::D$ for denoting subclasses. We remark that the exact semantic interpretation of such syntax might depend on the reasoning engine that is employed. A typical approach is to axiomatize :: and similar syntactical constructs in the underlying (non-monotonic) logic programming paradigm, and to regard classes and relations as logical terms, rather than as predicates. In contrast, the original semantics of F-Logic [11] was based on so-called "F-Structures," and employed classical monotonic negation. In order to avoid lengthy discussions on the intended semantics, we employ well-known Prolog syntax for the following examples, and merely remark that all of the programs could be adjusted for use in F-Logic reasoners such as OntoBroker, FLORA, or TRIPLE as well.

Besides the serialization in Prolog and F-Logic, the implementation also enables the user to serialize the translated rulebase in either RuleML 0.9[14] or alternatively in the proposed Semantic Web Rule Language SWRL [8] extension of OWL.

Note that SWRL as of now does not allow for the use of disjunctive head atoms in a rule. Therefore, the semantics of SWRL rules in KAON2 do not follow the standard at this point. In the standard, multiple atoms in the head of a rule should be interpreted as being conjunctive, but the serialization will print them meaning a disjunction. In order to avoid ambiguities, only SWRL rules with a single literal in the head should be used when interchanging SWRL files. The implementation will warn you, if you have rules with multiple head atoms.

For a concrete example of the transformation, consider the ontology given in Table 5 (top). The according trans-

[12] `http://kaon2.semanticweb.org`

[13] `http://owltools.ontoware.org`

[14] `http://www.ruleml.org/0.9/`

lation to Horn-logic is given in the middle and lower parts of the table. Let us first consider the middle part, which shows the rules directly created in the translation. Some of the rules clearly represent (part of) some $\mathcal{SHIQ}$-axiom, as is the case for "person(X) :– nosiblings(X)." and axiom (4). Other rules are obtained by more complicated reasoning steps, such as e.g. "parent(X) :– manychildren(X)." which is obtained from axioms (3) and (1). While such rules are still fairly self-explicatory, there is also a number of axioms that include predicates of the form $S_f(X, X_f)$ which do not appear in the original knowledge base. These predicates are introduced in the elimination of function symbols. Intuitively, $S_f(X, Y)$ holds iff $Y = f(X)$. However, the predicates S_f are only satisfied for a finite number of constants, since arbitrary application of functions is not needed and might even lead to undecidability. The exact number of additional function symbols may vary from case to case. Finally, two of the rules represent inconsistency constraints by means of the predicate inc as discussed in Section 3.3.

The rules at the bottom of Table 5 define various auxiliary predicates that are needed for the correctness of the translation. In order to restrict these definitions to a finite number of terms, we introduce a predicate O that specifies the individuals for which the program applies. In our case, these are just the individuals from the ABox. Using O, we define S_f as discussed above. Further, we introduce a predicate HU defining which terms are considered in the program, namely individuals from O and their immediate successors for each function symbol. The remaining rules yield the equality theory of Section 3.4, though restricted to the terms in HU.

The resulting program now allows us to conclude several ABox statements. For example, we can derive that "parent(Elaine)" and that "Sir Lancelot $\approx$ Lancelot du Lac." However, asking queries requires special care, since the generated program is not semantically equivalent to the original knowledge base. We discuss this aspect in the next section in greater detail.

5 Realized interoperability

The transformation of OWL DL into Horn logic programs described so far can be used to check satisfiability of the knowledge base in a logic programming system. In this section we describe which programs and rules can now safely be added to the output of the transformation without loosing soundness or completeness of reasoning. The following discussion affects querying as well, since asking a query is equivalent to adding the query (as a rule with an empty head) to the program, and checking satisfiability.

It has been remarked earlier that, in general, adding further axioms to the transformed program requires to invoke the transformation algorithm again. The reason is that axioms usually must be involved in the saturation step (3) of the transformation, as described in Section 3.1. Adding axioms after the transformation thus impairs the saturation, so that elimination of function symbols can yield incompleteness.

5.1 Adding ground facts

First, we discuss under which circumstances ground literals (positive or negative) can be added to the transformed knowledge base. Ground facts of the form $C(a)$, $\neg C(a)$, and $R(a, b)$ do not affect the saturation step of the transformation algorithm, since the associated inferences can also be drawn from the final logic program. Thus, one can generally disregard ABox axioms in the transformation and add them afterwards instead. However, as illustrated in Section 4, the transformed program needs to contain auxiliary axioms for each individual that occurs in the ABox. Thus, when adding axioms that introduce new individuals, one has to add an axiom of the form O(a) to the program as well.

Adding auxiliary facts O(a) corresponds to "registering" new individuals for reasoning with the program. This is crucial, since the generated Datalog program does only refer to the registered individuals. Indeed, every rule that is obtained by the transformation uses only variables that (finitely) depend on the predicate O. Thus, one can use the generated rules only to deduce facts about the known individuals, and reasoning for non-registered individuals is bound to be incomplete. As we will see in the next section, restriction to known individuals also ensures decidability.

The ability to add $\neg C(a)$ to the resulting disjunctive Datalog does of course refer to the usual first-order semantics of negation. In the logic programming setting, it is thus translated to $\leftarrow C(a)$, which can either be read as an integrity constraint that disallows $C(a)$, or as a query asking whether $C(a)$ is provable. As discussed in Section 3.3, the results derived in this situation by a Horn logic programming system fully agree with the semantics of first-order logic in both cases. One should be aware that this hinges upon the fact that one only asks for positive information: the given integrity constraint, when interpreted by the logic program, states that $C(a)$ cannot be true, not that it is false in a classical sense. Especially, negated ABox statements have nothing to do with the nonmonotonic negation (as failure) that is used in some logic programming systems.

The situation for facts of the form $\neg R(a, b)$ is more involved than for the above cases. Note that, in description logics, such statements are often not allowed in the ABox at all, and thus the lack of support for such negations does not affect the soundness or completeness of the transformation algorithm. Still, as observed in [15], adding $\neg R(a, b)$ to the knowledge base is unproblematic if the role R is *simple*, i.e., has no transitive subroles. As remarked in Section 3.1, the

TBox/RBox			ABox
(1)	Parent	$\equiv \exists$ hasChild.$\top$	hasChild(Elaine, Sir Lancelot)
(2)	Person	$\sqsubseteq \exists$ childOf.Person	noSiblings(Lancelot du Lac)
(3)	ManyChildren	$\sqsubseteq \geq 2$ hasChild.$\top$	childOf(Lancelot du Lac, Elaine)
(4)	NoSiblings	$\sqsubseteq$ Person $\sqcap \forall$ childOf.(≤ 1 hasChild.$\top$)	
(5)	childOf	$\equiv$ hasChild^{-1}	

person(X) :– nosiblings(X).
parent(X) :– haschild(X, Y).
haschild(Y, X) :– childof(X, Y).
haschild(X, X_{f2}) :– parent(X), $S_{f2}(X, X_{f2})$.
childof(X, X_{f3}) :– person(X), $S_{f3}(X, X_{f3})$.
person(X_{f3}) :– person(X), $S_{f3}(X, X_{f3})$.
parent(X) :– manychildren(X).
haschild(X, X_{f1}) :– manychildren(X), $S_{f1}(X, X_{f1})$.
haschild(X, X_{f0}) :– manychildren(X), $S_{f0}(X, X_{f0})$.
childof(Y, X) :– haschild(X, Y).

$Y_1 \approx Y_2$:– nosiblings(X), childof(X, Z), haschild(Z, Y_1), haschild(Z, Y_2).
inc :– manychildren(X), nosiblings(X_0), childof(X_0, X).
inc :– $X_{f1} \approx X_{f0}$, manychildren(X), $S_{f1}(X, X_{f1})$, $S_{f0}(X, X_{f0})$.

$S_f(X, f(X))$:– O(X). HU(X) :– O(X). HU($f(X)$) :– O(X). (for $f \in \{f_0, f_1, f_2, f_3\}$)

$X \approx X$:– HU(X).
$X \approx Y$:– $Y \approx X$, HU(X), HU(Y).
$X \approx Z$:– $X \approx Y, Y \approx Z$, HU(X), HU(Y), HU(Z).
$C(Y)$:– $C(X), X \approx Y$, HU(X), HU(Y).
(for $C \in$ {person, parent, manychildren, nosiblings})
$R(Y_1, Y_2)$:– $R(X_1, X_2), X_1 \approx Y_1, X_2 \approx Y_2$, HU($X_1$), HU($X_2$), HU($Y_1$), HU($X_2$).
(for $R \in$ {childof, haschild})

O(Elaine). O(Sir Lancelot). O(Lancelot du Lac).

Table 5. An example ontology in Horn-$\mathcal{SHIQ}$ (top), and its translation into Horn-logic, consisting of the translated rules (middle) and auxiliary axioms (bottom).

elimination of transitivity axioms performed by the algorithm does not yield a logically equivalent knowledge base. Moreover, even equisatisfiability is only ensured for formulae that regularly belong to $\mathcal{SHIQ}$.

Fortunately, the additional expressiveness of Horn logic allows us to recover transitivity axioms for the case of negative non-simple roles as well. Indeed, transitivity of R can simply be expressed by a rule $R(X, Z) \leftarrow R(X, Y) \wedge R(Y, Z)$.

As discussed in the following section, we could slightly extend this rule to make it *DL-safe* and ensure decidability. The addition of the above rule obviously is sound. On the other hand, attempting to remove all transitivity axioms in a preprocessing step and replacing them by axioms of the above form after the transformation might destroy completeness of the algorithm. The reason is that (the transformed) transitivity axioms play an important role during the saturation step, where one must draw enough consequences to justify the elimination of function symbols.

5.2 Complex rules and queries

Let us now turn to more complex rules and queries. Given the discussion in the previous section, it should be clear that all individuals that are added to a program must be "registered" via statements of the form O(a). Failing to do so generally results in incompleteness. If this is taken into account, arbitrary ground (Horn) rules can safely be introduced into the program. The situation for rules and queries that include variables is more complicated, and registering individuals is not sufficient in this case.

However, reasoning remains sound and complete if one considers only rules that are *DL-safe*. The intention is to restrict the scope of rules to those known individuals that occur in the (finite) Herbrand universe of the transformed program. More formally, we define a DL-safe rule as a function-free Horn rule $A \leftarrow B_1 \wedge \ldots \wedge B_n$, such that for every variable X that occurs in the rule, there is a non-DL atom B_i within which X occurs. A common way to achieve DL-safety is to add HU(X) for each variable occurring in the head of a rule, effectively restricting each rule to the Herbrand universe of the program.

It is also well known that the combination of OWL DL with arbitrary Horn-rules is undecidable [8]. Restricting to DL-safe rules thus does not only ensure completeness of reasoning, but is also needed to obtain a decidable formalism. Indeed, it is easy to see that DL-safe rules do not in-

troduce major termination problems, since they allow only for finitely many possible variable assignments. In fact, any DL-safe rule, and all of the generated program, could also be "unfolded" into finitely many ground rules, and treated as a propositional logic specification.

In addition to normal Horn-rules, many logic programming systems also support certain kinds of nonmonotonic negation operators. These can often be described as a kind of *negation as failure*, though the exact definition of "failure" may vary (e.g. in taking cyclic proofs into account or not). One might ask to what extent the generated program allows for a sound and complete interaction with such nonmonotonic formalisms. However, given that the formal semantics of such systems is often not defined with respect to first-order logic, it is not clear what "sound" or "complete" means in this setting. Clearly, nonmonotonic negation is not sound with respect to the semantics of negation in description logics, and thus cannot be used to query for such knowledge. On the other hand, nonmonotonic negation can still be used "on top" of the knowledge base, provided that the results are interpreted in an appropriate way. For example, the query $\leftarrow \mathsf{not}\, C(a)$ can safely be used to find out whether the logic programming system fails to construct a proof for $C(a)$, and this knowledge could be used by other modules of the logic program.

5.3 Termination

In addition to the question of semantic interoperability, one also has to take into account whether the logic programming system is actually able to work with the generated program. Even though the problem of determining satisfiability is decidable for the class of knowledge bases we consider, this does not mean that a particular system will actually decide all of the cases. Although the generated program could be used with almost any logic programming system, it turns out that simple SLD-resolution may run into loops when evaluating the generated programs. Since the program refers to finitely many individuals only, it is obvious that termination is not a problem in principle. Modern systems such as XSB Prolog usually employ *tabling* to detect loops (or use modified bottom-up strategies like OntoBroker), and this always suffices to obtain termination in our setting.

5.4 Offline processing

We have seen that certain facts, including ABox assertions and negated ABox assertions for named classes, can be added to the knowledge base after the translation performed by the KAON*2* translation algorithms. This feature can be put to use for practical query answering, as it allows to translate the knowledge base offline, which means that the KAON*2* algorithms do not have to be invoked again once knowledge of a basic form is added or when the system is queried. The benefit from this offline translation is apparent from the involved worst-case complexities: The KAON*2* translation algorithms are ExpTime, while the resulting Datalog is polynomial.

The basic functionality of our approach is thus as follows: Given a Horn-$\mathcal{SHIQ}$ knowledge base, it is transformed offline into Datalog. ABox assertions for named classes can then be added to the Datalog knowledge base as they become known. For querying, we allow only the retrieval of instances of named classes, as this can be performed without touching the Datalog knowledge base.

In principle however, it is possible to query for instances of certain complex classes or TBox knowledge, however this involves the addition of some OWL axiom to the original Horn-$\mathcal{SHIQ}$ knowledge base, which in turn necessitates to invoke the KAON*2* translation algorithm again. We think that this process can be enhanced by using incremental algorithms, but this remains to be investigated.

6 Conclusions

Employing the transformation algorithm of KAON*2*, we gave a detailed description of how a considerable fragment of OWL DL can be processed within logic programming systems. To this end, we derived an enhanced characterization of Horn-$\mathcal{SHIQ}$, the description logic for which this translation is possible, and explained how the generated Datalog programs can be used in a standard logic programming paradigm without sacrificing soundness or completeness of reasoning. The primary contribution of our study thus is to clarify to which extent KAON*2* enables semantic interoperability between the ontology modeling paradigms of description logics and logic programming, and how this interoperability can be realized in practice. Nonetheless, the described procedure may actually find practical applications within research projects such as SmartWeb.[15]

Our contribution leaves various open questions to be investigated in the future. Can the KAON*2* transformation algorithms be used for merging OWL and F-Logic databases? Can they possibly contribute to establishing powerful integrated or hybrid systems based both on the logic programming *and* on the description logic tradition? Can practical systems for large scale applications be developed where different ontology language paradigms can be dealt with in an interoperable way?

The sophisticated KAON*2* algorithms significantly enlarge the class of $\mathcal{SHIQ}$ ontologies processable in a logic programming system. Although this extended scope is reflected in an increased worst-case complexity and rules out the possibility of a semantically equivalent transformation,

[15] http://www.smartweb-project.de

it still allows for nontrivial semantic interactions between the two paradigms. Such interaction does not only yield ways to transfer knowledge between heterogeneous systems, but also, as in the case of DL-safe rules, can encompass future extensions of the current ontology modeling languages.

References

[1] J. Angele, H. Boley, J. de Bruijn, D. Fensel, P. Hitzler, M. Kifer, R. Krummenacher, H. Lausen, A. Polleres, and R. Studer. Web Rule Language (WRL). W3C Member Submission 09 Sept. 2005, `http://www.w3.org/Submission/2005/SUBM-WRL-20050909/`, 2005.

[2] T. Eiter, G. Ianni, R. Schindlauer, and H. Tompits. Effective integration of declarative rules with external evaluations for semantic-web reasoning. In Y. Sure and J. Domingue, editors, *Proc. of Third European Semantic Web Conference (ESWC2004), Budva, Montenegro, June 2006*, volume 4011 of *Lecture Notes in Computer Science*, pages 273–287. Springer, 2006.

[3] T. Eiter, T. Lukasiewicz, R. Schindlauer, and H. Tompits. Combining answer set programming with description logics for the semantic web. In D. Dubois, C. A. Welty, and M.-A. Williams, editors, *Proc. of Ninth Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR2004), Whistler, Canada, June 2004*, pages 141–151. AAAI Press, 2004.

[4] B. Grosof, I. Horroks, R. Volz, and S. Decker. Description logic programs: Combining logic programs with description logics. In *Proc. of WWW 2003, Budapest, Hungary, May 2003*, pages 48–57. ACM, 2003.

[5] P. Hitzler, P. Haase, M. Krötzsch, Y. Sure, and R. Studer. DLP isn't so bad after all. In *Proc. of the WS OWL – Experiences and Directions, Galway, Ireland, November 2005*, 2005.

[6] P. Hitzler, R. Studer, and Y. Sure. Description logic programs: A practical choice for the modelling of ontologies. In *1st WS on Formal Ontologies meet Industry, FOMI'05*, 2005.

[7] I. Horrocks, B. Parsia, P. Patel-Schneider, and J. Handler. Semantic web architecture: Stack or two towers? In F. Fages and S. Soliman, editors, *Proc. of Principles and Practice of Semantic Web Reasoning. 3rd Int. WS, PPSWR 2005, Dagstuhl Castle, Germany, Sept. 2005*, volume 3703 of *Lecture Notes in Computer Science*, pages 37–41. Springer, Berlin, 2005.

[8] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A Semantic Web Rule Language combining OWL and RuleML. W3C Member Submission 21 May 2004, `http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/`, 2004.

[9] U. Hustadt, B. Motik, and U. Sattler. Reducing $\mathcal{SHIQ}$ description logic to disjunctive datalog programs. In D. Dubois, C. A. Welty, and M.-A. Williams, editors, *Proc. of the 9th Int. Conf. on Knowledge Representation and Reasoning (KR2004)*, pages 152–162, 2004.

[10] U. Hustadt, B. Motik, and U. Sattler. Data complexity of reasoning in very expressive description logics. In *Proc. 19th Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 466–471, 2005.

[11] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the Association for Computing Machinery*, 42:741–843, 1995.

[12] M. Krötzsch, S. Rudolph, and P. Hitzler. On the complexity of Horn description logics. Technical report, Institute AIFB, Universität Karlsruhe, 2006.

[13] J. W. Lloyd. *Foundations of Logic Programming*. Springer, Berlin, 1988.

[14] D. McGuinness and F. v. Harmelen. OWL Web Ontology Language Overview, 2004. W3C Recommendation 10 Feb. 2004, `http://www.w3.org/TR/owl-features/`.

[15] B. Motik. *Reasoning in Description Logics using Resolution and Deductive Databases*. PhD thesis, Universität Karlsruhe (TH), Germany, 2006.

[16] B. Motik, D. Vrandečić, P. Hitzler, Y. Sure, and R. Studer. dlpconvert – converting OWL DLP statements to logic programs. In *European Semantic Web Conference 2005 Demos and Posters*, May 2005. System available at `http://owltools.ontoware.org/`.

[17] R. Rosati. DL+log: Tight integration of description logics and disjunctive datalog. In P. Doherty, J. Mylopoulos, and C. A. Welty, editors, *Proc. of Tenth Int. Conf. on Principles of Knowledge Representation and Reasoning (KR2006), Lake District of the United Kingdom, June 2006*, pages 68–78. AAAI Press, 2006.

[18] J. S. Schlipf. The expressive powers of the logic programming semantics. *Journal of Computer and System Sciences*, 51:64–86, 1995.

[19] M. Sintek and S. Decker. TRIPLE – A query, inference, and transformation language for the semantic web. In *Proc. of the Int. Semantic Web Conf. (ISWC02), Sardinia, Italy*, 2002.

[20] R. Volz. *Web Ontology Reasoning with Logic Databases*. PhD thesis, Universität Karlsruhe (TH), Germany, 2004.