Any-World Access to OWL from Prolog

Tobias Matzner and Pascal Hitzler

Institute AIFB, Universität Karlsruhe, Germany
{tobias.matzner, hitzler}@aifb.uni-karlsruhe.de

Abstract. The W3C standard OWL provides a decidable language for representing ontologies. While its use is rapidly spreading, efforts are being made by researchers worldwide to augment OWL with additional expressive features or by interlacing it with other forms of knowledge representation, in order to make it applicable for even further purposes. In this paper, we integrate OWL with one of the most successful and most widely used forms of knowledge representation, namely Prolog, and present a hybrid approach which layers Prolog on top of OWL in such a way that the open-world semantics of OWL becomes directly accessible within the Prolog system.

1 Introduction

The Web Ontology Language OWL has been recommended by the W3C in 2004 for the representation of ontologies, and its usage is spreading rapidly ever since. One of the design issues for OWL has been that it is decidable and based on the open world assumption, and these two properties – which are both inherited from description logics – have served it well in the last two years.

However, with these design decisions come also some drawbacks as they limit expressiveness of OWL in ways which make working with it cumbersome at times. Even more, due to decidability of the language some things cannot be expressed at all in OWL. Efforts are therefore under way to extend OWL with more expressive features, and there is a growing body of work with proposals and studies how to do this best.

The corresponding research can roughly be classified into two different approaches. The first approach deals with extensions of OWL while adhering as much as possible to the conceptual frame of mind spanned by description logic research. The second approach is based on establishing hybrid systems which combine OWL with other established knowledge representation formalisms in such a way that either approach is encompassed in full, possibly using two different reasoning engines, but allowing for information flow between the subsystems. The work which we present in this paper is of the hybrid kind.

The particular integration which we report on, is based on the following rationales.

– OWL has not been designed to be a stand-alone programming language. OWL ontologies should rather be viewed as declarative knowledge bases,

which require programming in some other language for accessing the knowledge and further processing it. It is a natural choice to use a logic-based declarative programming language for this purpose.
– One of the most requested-for extensions of OWL is the ability to formulate rules, in some established rules language.
– It becomes more and more apparent that closed-world features are required alongside the open-world character of OWL.

Our hybrid system addresses the formulated needs by interlacing OWL with one of the most prominent and historic approaches to logic-based knowledge representation, namely with Prolog. Our system layers Prolog on top of OWL by allowing the querying of OWL ontologies via a standard OWL reasoner. A tight integration is achieved by interpreting the answers given by the OWL reasoner in an open-world fashion, and by processing this answer within Prolog in the same open-world fashion. This is achieved by means of the so-called any-world semantics due to Loyer and Straccia [1].

Technically speaking, the integration is achieved via a hybrid semantics for a language which incorporates calls to an OWL reasoner into standard logic programming. This hybrid semantics is based on the any-world semantics. Algorithmization and an implementation of the approach is provided by means of a transformation of logic programs under the any-world semantics into standard Prolog, in this case realised using SWI-Prolog.

Besides the aforementioned rationales for our approach, we thus arrive at a system with the following additional features.

– Modularity: The user can develop its programs based on Prolog programming and need not deal with the evaluation of OWL-based reasoning and knowledge. It is possible to offer restricted or controlled access to third party knowledge-bases without problems.
– Maturity: We incorporate the KAON2 reasoner and thus offer the performance of a state-of-the-art DL-reasoner to the logic programming world. The logic programming environment can be handled with little more than basic Prolog knowledge.
– Conformity with standards: Available OWL knowledge bases can be used directly. As we do not need one big formal system comprising both approaches, these can be used with no or only little maintenance to do.
– Bridge between ontology language paradigms: One of the most prominent alternatives to OWL for ontology representation is F-Logic [2, 3], which can be used both as an ontology language and as a programming language. As F-Logic in its basic form is basically Prolog extended with further syntactic features, our approach can be used directly for realising a hybrid OWL/F-Logic system.

The structure of the paper is as follows. In Section 2, we review the basic facts we need about the any-world semantics and about OWL in order to make this paper relatively self-contained. In Section 3, we prove a theorem which gives the formal rationale for our algorithmisation. In Section 4 we discuss the

implemented system which we provide. In Section 5 we give an extended example which shows the possibilities of our approach. In Section 6 we discuss related work, and we conclude in Section 7.

Acknowledgements We gratefully acknowledge support by the German Ministry for Education and Research under the SmartWeb project grant 01 IMD01 B, by the European Commission under the NeOn project IST-2006-027595, and by the Deutsche Forschungsgemeinschaft under the ReaSem project. We would also like to thank the members of the OntoLoRe group at AIFB Karlsruhe, and in particular Markus Krötzsch, for helpful discussions.

2 Preliminaries

2.1 The Any-World semantics

We review the any-world semantics due to Loyer and Straccia [1] in some details as it is crucial for understanding our work.

Bilattices The any-world semantics is based on a truth-space which is a so-called bilattice [4]. This is a potent mathematical structure which particularly provides two partial orders, which permit to represent (logical) truth and the knowledge contained in these truth-values separately.

Formally, a *lattice* $\langle L, \leq \rangle$ is a non-empty set L with a partial order $\leq$, where each subset of L containing two elements has a supremum and infimum regarding $\leq$ (also known as *meet* and *join*). It is a *complete lattice* iff every subset has supremum and infimum regarding $\leq$. We write $x < y$ for $x \leq y$ and $x \neq y$ where $x, y \in L$.

A *bilattice* $\langle B, \leq_t, \leq_k \rangle$ is a non-empty set B with two partial orders, the *truth-order* $\leq_t$ and the *knowledge-order* $\leq_k$, both of which give B the structure of a complete lattice. Due to completeness, the greatest and least element regarding either of the orders always exists and is unique [4]. The greatest element regarding $\leq_t$ is denoted `true`, the least element `false`. Regarding $\leq_k$, the greatest element is $\top$, the least $\bot$. Meet and join under $\leq_t$ which are denoted $\wedge$ and $\vee$, correspond to the well-known two-valued conjunction and disjunction regarding the values `true` and `false`. Under $\leq_k$ meet and join are denoted $\otimes$ and $\oplus$, where $x \otimes y$ extracts the maximum knowledge that is expressed both in x and y whereas $x \oplus y$ unites the knowledge of x and y. Our approach is particularly based on the smallest non-trivial bilattice known as $\mathcal{FOUR}$ [5] which is depicted in Figure 2.1. Indeed, although bilattice-based semantics is generally formulated for arbitrary bilattices, $\mathcal{FOUR}$ is currently the only such lattice of practical relevance, and will entirely suffice for our purposes.

An operator $\bullet$ on a lattice is called *monotone* when $x_1 \leq y_1$ and $x_2 \leq y_2$ implies $x_1 \bullet x_2 \leq y_1 \bullet y_2$. We suppose for all bilattices here considered that all of the operators $\wedge, \vee, \otimes, \oplus$ are monotone w.r.t. both the knowledge- and the truth-order; this is called the *infinitary interlacing condition*. We furthermore

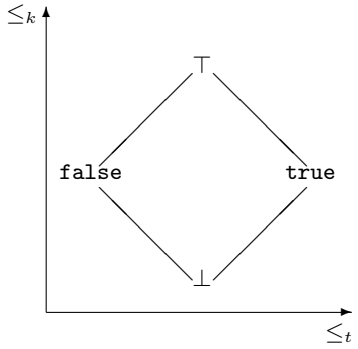

Fig. 1. The bilattice $\mathcal{FOUR}$

assume that all bilattices are *infinitary distributive* i.e. that all distributive laws connecting the aforementioned lattice operators hold. Finally, we assume that all lattices have a *negation*, which is an operator denoted $\neg$ that inverses the truth order, does not inflict the knowledge order and satisfies $\neg\neg x = x$. These assumptions are standard and generally known to be unproblematic in a logic programming context.

Logic programs We extend logic programs from the common case and include not only connectives for disjunction, conjunction and negation but for all the operators of a bilattice: $\wedge, \vee, \otimes, \oplus$ and $\neg$. So the knowledge order and its operators are not only a tool of analysis and semantics as used for example in [6] but can be used explicitly to determine how the program treats information from the perspective of knowledge. A logic program is based on a set $\mathcal{P}$ of predicates, $\mathcal{V}$ of variables, $\mathcal{C}$ of constants and $\mathcal{F}$ of functions. A *term* is either an element of $\mathcal{V}$ or $\mathcal{C}$ or of the form $f(t_1, \ldots, t_n)$ where $f \in \mathcal{F}$ and all $t_1, \ldots, t_n$ are terms. The *ground terms* forming the *Herbrand universe* are all the terms that can be built from elements of $\mathcal{C}$ and $\mathcal{F}$. An *atom* is of the form $p(t_1, \ldots, t_m)$ where $p \in \mathcal{P}$ and all $t_1, \ldots, t_m$ are terms. The *ground atoms* forming the *Herbrand base* are all the atoms that can be built from the Herbrand universe. A *literal* is of the form A or $\neg A$ where A is an atom. Furthermore we allow the elements of the bilattice as literals. A *formula* is either any literal, or of the form $\varphi_1 \bullet \varphi_2$ where φ_1 and φ_2 are formulas and $\bullet$ is one of the four lattice operators $\wedge, \vee, \otimes, \oplus$, or one of the expressions $\forall\varphi$ respectively $\exists\varphi$ where φ is a formula. A *rule* is of the form $p(x_1, \ldots, x_m) \leftarrow \varphi(x_1, \ldots, x_m)$ where $p \in \mathcal{P}$, $x_1, \ldots, x_m \in \mathcal{V}$ and φ is a formula. We call p the *head* and φ the *body* of the rule. We suppose that the free variables in φ are among $\{x_1, \ldots, x_m\}$ and are universally quantified. A *logic program* P is a finite set of rules. Not allowing terms in the heads of rules is not a restriction, e.g. the rules (taken from [1]):

$$p(s(x)) \leftarrow p(x)$$
$$p(0) \leftarrow \texttt{true}$$

can be rewritten (using a predicate eq defining equality) as:

$$p(y) \leftarrow \exists x(eq(y, s(x)) \wedge p(x))$$
$$p(x) \leftarrow eq(x, 0)$$

With $ground(P)$ we denote all ground instances of members of P over the Herbrand universe.

Interpretations of logic programs Let B be a bilattice. An *interpretation* of a logic program on B is a mapping I from ground atoms to members of B. It is extended to formulas as follows: $I(b) = b$ where $b \in B$; $I(\varphi_1 \bullet \varphi_2) = I(\varphi_1) \bullet I(\varphi_2)$ where φ_1, φ_2 are formulas and $\bullet$ is one of the operators $\wedge, \vee, \otimes, \oplus$; $I(\neg\varphi) = \neg I(\varphi)$; $I(\exists x\varphi(x)) = \bigvee\{I(\varphi(t))|t \text{ is a ground term}\}$ and finally $I(\forall x\varphi(x)) = \bigwedge\{I(\varphi(t))|t \text{ is a ground term}\}$. The partial orders of the bilattice are point-wise extended to interpretations: $I_1 \leq_t I_2$ iff $I_1(A) \leq_t I_2(A)$ for all ground atoms A. The extension for $\leq_k$ is analogous. Given two interpretations I_1, I_2 we define $(I_1 \bullet I_2)(\varphi) = I_1(\varphi) \bullet I_2(\varphi)$ where $\bullet$ is a lattice operator and φ a formula. Thus the space of all possible interpretations on a bilattice constitutes an infinitary interlaced and distributive bilattice as well. An interpretation I is a *model* of a logic program P iff $I(A) = I(\varphi)$ for all rules $A \leftarrow \varphi$ in P.

Semantics The semantics is defined via the fixed point of a monotone operator similarly to the well known Kripke-Kleene [7, 8] or well-founded [9] semantics. In fact the any-world semantics used here is a generalization of the well-founded semantics. The central idea of the any-world semantics is to overcome the limitations of both the open and the closed world as default assumption. Instead an arbitrary interpretation H called the *hypothesis* is used as default assumption, i.e. the value $H(A)$ is the default value for the atom A. From this point of view, the open world assumption corresponds to the hypothesis $H(A) = \bot$ for all atoms A, we call this hypothesis $H_\bot$. The closed world assumption can be modelled by $H(A) = \texttt{false}$ for all atoms A, this hypothesis is denoted $H_\texttt{f}$. Now the information of the program is combined with knowledge extracted from the hypothesis used. To gather information from the program we use the well known *immediate consequence operator* $\Phi_P(I)(A) = I(\varphi)$ where $A \leftarrow \varphi$ is a rule in P. Now we want to augment the interpretation I with the information from a hypothesis H. This is done similarly to the use of the unfounded set in the well-founded semantics. From a knowledge point of view, the unfounded set is the amount of information contributed to the semantics by the closed world assumption. This concept now is generalized to arbitrary hypotheses H. We usually cannot use all the information of H. Instead we want to extract the maximum knowledge of H, expressed as an interpretation J, so that the assumed knowledge J is entailed by the program w.r.t. the augmented interpretation $I \oplus J$, i.e. we want to make sure that $J(A) \leq_k \Phi_P(I \oplus J)(A)$. This idea is modelled using the so called *safe interpretations*. An interpretation J is *safe* w.r.t. a logic program P, an interpretation I and a hypothesis H if $J \leq_k H$ and

$J \leq_k \Phi_P(I \oplus J)$. The *support* provided by H to P and I is the greatest (on the knowledge order) safe interpretation w.r.t. P, I and H. It is denoted $s_P^H(I)$. Note that this particularily entails that the support is always smaller than the hypothesis.

In order to simplify the treatment of logic programs using fixed-point semantics, we introduce the transformed program P^*. Given a logic program P and a hypothesis H the program P^* contains the following rules:

- $A \leftarrow \varphi_1 \vee \cdots \vee \varphi_n$ if $A \leftarrow \varphi_1, \ldots, A \leftarrow \varphi_n$ are all rules in $ground(P)$ with the head A.
- $A \leftarrow H(A)$ if A is not the head of any rule in P.

The second part enforces that for any atom that is not assigned a truth-value by a rule in the program, it is given its value according to the default assumption, i.e. the hypothesis.

Now we define the operator $\tilde{\Pi}_P^H(I) = \Phi_P(I) \oplus s_P^H(I)$ which works on P^*. The fixed points of $\tilde{\Pi}_P^H$ are called the *H-founded models* of P. In [1] it is shown that the support operator $s_P^H(I)$ is monotone in I and H w.r.t. the knowledge order. Furthermore also Φ_P is monotone w.r.t. the knowledge order [6]. By the infinitary interlacing condition, motonicity of $\tilde{\Pi}_P^H$ is guaranteed. So by the well known Knaster-Tarski theorem [10], there is always a (unique) least H-founded model for any logic program, which can be obtained as the least upper bound of the transfinite sequence $(\tilde{\Pi}_P^H \uparrow \alpha)_\alpha$, where α ranges over ordinals, $\tilde{\Pi}_P^H \uparrow 0$ is the least interpretation, $\tilde{\Pi}_P^H \uparrow \alpha + 1 = \tilde{\Pi}_P^H(\tilde{\Pi}_P^H \uparrow \alpha)$ for all α, and $\tilde{\Pi}_P^H \uparrow \alpha = \sup\{\tilde{\Pi}_P^H \uparrow \beta : \beta < \alpha\}$ for limit ordinals α.

The key feature of the any-world semantics is the flexibility of the default assumption. Particularly using a hypothesis that maps ground atoms to the set $\{\texttt{false}, \bot\}$ it is possible to mix closed- and open-world based information, whereon our hybrid semantics relies. It also includes several well known semantics. Using $H_{\texttt{f}}$, the H-founded model is the well-founded model [1]. Let H_{KK} be the interpretation that maps all atoms that are the head of a rule in a given program P to $\bot$, all the other atoms to `false`. Using this hypothesis, the H-founded model of P is its Kripke-Kleene-model [1]. This reflects that the Kripke-Kleene semantics uses only the immediate-consequence operator Φ_P and consequently the support part in $\tilde{\Pi}_P^H$ is reduced to $\bot$ by the assignment of $\bot$ to all rule heads. (Recall that the support is always smaller than the hypothesis on the knowledge order). However the Kripke-Kleene semantics is based on the closed world assumption. This is manifested in the hypothesis mapping the other atoms to `false`. The fact that the hypothesis affects only those atoms will be used later for the hybrid semantics of our system.

2.2 Description logics

The description logics part of our hybrid system uses the KAON2 OWL DL reasoner [11].[1] Our approach, however, is independent of the specific reasoner

[1] See also http://kaon2.semanticweb.org

used, and can indeed be used with any reasoning system based on the open world assumption. OWL DL is based on the description logic $\mathcal{SHOIN}$(D) [12], but for the purpose of our exhibition we will not need to give many details about OWL DL. It shall suffice to recall that OWL DL allows to specify axioms describing the subsumption relation between complex concepts C and D, written $C \sqsubseteq D$. The (complex) concepts themselves are composed by means of primitive (or atomic) concepts, logical and other connectives, individuals which correspond to logical constants, and roles which describe relationships between individuals. It is also possible to specify that some individual a belongs to a class C, written $C(a)$, or to explicitly state that two individuals a and b are connected by a role R, written $R(a, b)$. The special concepts $\top$ and $\bot$, respectively, are defined as containing all individuals respectively no individual. OWL DL is given an open-world semantics e.g. by mapping it into first-order logic with equality.

Given a set of OWL DL axioms, called an *ontology*, it is possible to derive logical consequences from it by means of well-established algorithms. The most basic inference tasks are

- checking whether an ontology is satisfiable (i.e. logically consistent),
- checking whether a concept C subsumes a concept D, i.e. whether $C \sqsubseteq D$ is a logical consequence,
- checking whether a concept C is satisfiable, i.e. whether there is a model of the knowledge base in which the extension of C is non-empty, and
- checking whether an individual a is contained in a concept C, i.e. whether $C(a)$ is a logical consequence.

3 A program transformation for algorithmising the any-world semantics

We provide an extension for Prolog which implements an any-world logic based on $\mathcal{FOUR}$ and hypotheses that map into $\{\texttt{false}, \bot\}$. The implementation is based on Theorem 1 below, which acts as a bridge between the any-world semantics and Prolog.

Before we provide the theorem, let us define the specific type of hypotheses which we need for our purposes. Recall that the hypothesis $H_{\mathtt{f}}$ corresponds to the closed world assumption, while $H_{\bot}$ can be interpreted as an open world semantics. Consequently, the hypotheses of interest are a mix between these two.

Definition 1. *Given a logic program P we define the set of hypotheses* KKS *to be the set of all interpretations that map an atom A to $\bot$ when A is the head of a rule in P, and to either $\bot$ or* `false` *otherwise.*

Note that H_{KK} is in KKS for all programs P.

Theorem 1. *Given a logic program P and a hypothesis $H_{\mathrm{KKS}} \in$ KKS, there exists a program transformation $T^{H_{\mathrm{KKS}}}$ such that the H-founded model of P under H_{KKS} is the same as the H-founded model of $T^{H_{\mathrm{KKS}}}(P)$ under H_{KK}.*

The proof is based on the possibility to add the default assumption chosen as rules of the form $A \leftarrow H(A)$ to the program, such that the resulting program does not have any atoms that are not head of a rule. When evaluated under H_{KK}, accordingly the default assumption `false` is not used for any atom. The assumption $\bot$ for atoms that are heads of a rule, i.e. all atoms, is overridden by the rule $A \leftarrow$ `false` should it exist, as `false` $\oplus \bot =$ `false`. We first prove the following:

Lemma 1. *Let P be a logic program and H_{KKS} a hypothesis from* KKS*. Then $s_P^{H_{KKS}}(I) = H_{KKS}$ for any interpretation I.*

Proof. For the following proof we write $s_P^{H_{KKS}}(A)$ for $s_P^{H_{KKS}}(I)(A)$ as the choice of interpretation is without effect. For an atom A that is not the head of any rule there are two possibilities: (1) If $H_{KKS}(A) = \bot$, then the fact that the support is always smaller than the hypothesis on the knowledge-order requires $s_P^{H_{KKS}}(A) = \bot$. (2) If $H_{KKS}(A) =$ `false`, then the rule $A \leftarrow$ `false` is in P^*. As the support is a safe interpretation we require $s_P^{H_{KKS}}(A) \leq_k \Phi_P(I \oplus s_P^{H_{KKS}})(A)$. This now becomes $s_P^{H_{KKS}}(A) \leq_k I(\texttt{false}) \oplus s_P^{H_{KKS}}(\texttt{false}) =$ `false`. As the support is the largest safe interpretation on the knowledge-order we have $s_P^{H_{KKS}}(A) =$ `false`. Consider now an atom A that is head of a rule in P. Using again that the support is a safe interpretation and thus smaller (in the knowledge order) than the hypothesis, we have that $s_P^{H_{KKS}}(A) = \bot = H_{KKS}$. □

Now we are ready to prove the theorem:

Proof (of Theorem 1). We use the notation of the theorem. Furthermore we let $P' = T^{H_{KKS}}(P)$. We now show that the operators $\tilde{\Pi}_P^{H_{KKS}}$ and $\tilde{\Pi}_{P'}^{H_{KK}}$ have the same result on every step of their iteration. As the operator $\tilde{\Pi}$ is defined on P^*, for the evaluation of $\tilde{\Pi}_P^{H_{KKS}}$ P^* is constructed from P under the hypothesis H_{KKS}. In this process the same rules are added as when applying $T^{H_{KKS}}$ to P by definition of T. So P' and P^* constructed under the hypothesis H_{KKS} are identical. For the evaluation of $\tilde{\Pi}_{P'}^{H_{KK}}$ the program P'^* is constructed under the hypothesis H_{KK}. Since in P' all atoms are head of rule, the hypothesis H_{KK} has no influence on P'^*. So we have $P^* = P'^*$, thus $\tilde{\Pi}_P^{H_{KKS}}$ and $\tilde{\Pi}_{P'}^{H_{KK}}$ work on the same program.

Now consider an arbitrary iteration step α:

$$(\tilde{\Pi}_{P'}^{H_{KK}} \uparrow \alpha + 1)(A) = (\tilde{\Pi}_{P'}^{H_{KK}} \uparrow \alpha)(\varphi) \oplus s_{P'}^{H_{KK}}(\tilde{\Pi}_{P'}^{H_{KK}} \uparrow \alpha)(A).$$

We have $H_{KK}(A) = \bot$ for all atoms A in P' as all atoms are the head of a rule after the transformation. By Lemma 1 the support is equal to the hypothesis (note that H_{KK} is in KKS) and so the remaining formula is

$$(\tilde{\Pi}_{P'}^{H_{KK}} \uparrow \alpha + 1)(A) = (\tilde{\Pi}_{P'}^{H_{KK}} \uparrow \alpha)(\varphi). \tag{1}$$

Consider now the operator

$$(\tilde{\Pi}_P^{H_{KKS}} \uparrow \alpha + 1)(A) = (\tilde{\Pi}_P^{H_{KKS}} \uparrow \alpha)(\varphi) \oplus s_P^{H_{KKS}}(\tilde{\Pi}_P^{H_{KKS}} \uparrow \alpha)(A).$$

Again by Lemma 1 we have that $H^{H_{KKS}}(A) = s_P^{H_{KKS}}(\tilde{\Pi}_P^{H_{KKS}} \uparrow \alpha)(A)$. For atoms that are head of a rule in P we obtain

$$(\tilde{\Pi}_P^{H_{KKS}} \uparrow \alpha + 1)(A) = (\tilde{\Pi}_P^{H_{KKS}} \uparrow \alpha)(\varphi). \tag{2}$$

As P and P' contain the same rules, the operators in (1) and (2) yield the same results. For atoms that are not the head of a rule we know that either $H_{KKS}(A) = \bot$ or $H_{KKS}(A) = \texttt{false}$. The following argument is analogous for both cases, we consider the latter. If $H_{KKS}(A) = \texttt{false}$, then there is a rule $A \leftarrow \texttt{false}$ in P^* and accordingly also in P'. So we have $(\tilde{\Pi}_P^{H_{KKS}} \uparrow \alpha)(\varphi) = s_P^{H_{KKS}}(\tilde{\Pi}_P^{H^{KKS}} \uparrow \alpha)(A) = \texttt{false}$ as well as $(\tilde{\Pi}_{P'}^{H^{KK}} \uparrow \alpha)(\varphi) = \texttt{false}$. So both operators give the same result. □

So we have the possibility to deal with hypotheses mixing open- and closed-world assumption while we only need to compute the Kripke-Kleene semantics.

4 Implementation

In order to arrive at its least fixed point, i.e. at the Kripke-Kleene semantics, Φ_P may need as many as Church-Kleene ω_1 steps. Indeed the Kripke-Kleene semantics is Π_1^1-complete [13] and thus not even semi-decidable. This means that a sound and complete implementation of the Kripke-Kleene semantics cannot be provided for theoretical reasons.

However, the Kripke-Kleene semantics was originally conceived as a declarative semantics which captures the essence of the Prolog procedural semantics, and indeed they are strongly related, as shown e.g. in [14]. For practical purposes, it thus suffices to view Prolog as an approximate implementation of the Kripke-Kleene semantics.

We therefore provide a library that permits using the logic $\mathcal{FOUR}$ with all corresponding lattice operations $\oplus, \otimes, \wedge, \vee$ and $\neg$ in Prolog. The user can write programs in a Prolog-like syntax, which is then compiled to SWI-Prolog[2] such that each predicate is augmented with an additional parameter, which carries the truth-value. A predicate $\mathsf{p}(\mathsf{t_1}, \ldots, \mathsf{t_n}, \mathsf{TV})$ is then deducible in Prolog if $\mathsf{p}(\mathsf{t_1}, \ldots, \mathsf{t_n})$ has the truth-value TV.

Within this framework, we offer special atoms, so called *DL-atoms*, that are not evaluated according to the logic programming semantics but by querying the DL-reasoner KAON2. They have the form $dl_q(\mathrm{p}_q)$ where q is a query and p_q the respective vector of parameters. The queries we offer are *subsumes, unsatisfiable* and *disjoint* regarding concepts and *has_role* regarding roles.

Usually queries to a DL-reasoner have two possible answers: the queried information is either demonstrable or not. However, if the answer is negative, then two cases are possible: Either the negation of the query is demonstrable, or the negation of the query is also not demonstrable. In the first case, the refutation of the query is much stronger than in the second.

[2] http://www.swi-prolog.org

In order to give an example, consider the knowledge base specified by the following axioms.

$$unicorn \sqsubseteq appears\_in\_novels$$
$$horned\_animal \sqsubseteq animal$$

When queried whether $unicorn \sqsubseteq horned\_animal$ holds, the reasoner responds with `No`, which is entirely appropriate as the knowledge base does not allow to derive any knowledge about the relationship between *unicorn* and *horned_animal*, i.e. the relationship $unicorn \sqsubseteq horned\_animal$ can neither be confirmed nor refuted.

Consider now the situation that the knowledge base contains the following additional axioms, where the second describes the assertion that the concepts *unicorn* and *phantasy_animal* are extensionally disjoint.

$$unicorn \sqsubseteq phantasy\_animal$$
$$animal \sqcap phantasy\_animal \sqsubseteq \bot$$

When now queried whether $unicorn \sqsubseteq horned\_animal$ holds, the reasoner again responds with `No`, which is entirely appropriate as the knowledge base implies that *unicorn* and *horned_animal* are in fact extensionally disjoint. The situation compared to the first situation, however, is very different: The first knowledge base did not specify anything about the relation between *unicorn* and *horned_animal*, while the second knowledge base strongly refutes the subsumption relation.

Our framework provides the means to distinguish between these situations by means of a different choice of truth values. In the first situation, the resulting truth value must be $\bot$, while in the second it must be `false`. Technically, we realise this in such a way that each query to KAON2 results in two calls to the reasoner allowing to retrieve more detailed information. For the atom $dl_{subsumes}(C, D)$, the first query to the reasoner asks for $C \sqsubseteq D$. Given a positive answer, we know that this is demonstrable, thus the DL-atom is evaluated as `true`. When the answer is negative, there are, however, two cases possible: $C \sqsubseteq D$ might be satisfiable, but not formally implied by the knowledge base. In this case the DL-atom should have the value $\bot$ i.e. unknown. On the other hand it is possible that the information in the knowledge-base makes $C \sqsubseteq D$ impossible. Then the DL-atom should be assigned `false`. This is done by the second query, which asks whether $KB \cup \{C \sqsubseteq D\}$ is satisfiable, where KB is the knowledge-base. We summarize the query in the following table.

result of the query: $C \sqsubseteq D$	result of the query: Is $KB \cup \{C \sqsubseteq D\}$ satisfiable?	value of $dl_{subsumes}(C, D)$
yes	–	`true`
no	yes	$\bot$
no	no	`false`

Note that the queries are executed consecutively, i.e. the second query is only performed if the first returned false.

A useful perspective on this is the following: $C \sqsubseteq D$ results in `true` if it holds in *all* models of the knowledge base. It results in `false` if it holds in *none* of the models of the knowledge base. And it results in $\bot$ if it holds in some, but not all, models of the knowledge base.

The question of the satisfiability of a concept is reducible to subsumption: a concept C is satisfiable iff $C \sqsubseteq \bot$ does not hold, i.e. if there is *some* model in which the extension of C is non-empty. This situation is best understood by considering *unsatisfiabilitiy* of a concept instead of satisfiability, as this allows us to use exactly the argumentation used above: A concept is unsatisfiable if it is extensionally empty in *all* models of the knowledge base. Similarly to the case of subsumption, we arrive at the execution detailed in the following table.

result of the query: $C \sqsubseteq \bot$	result of the query: Is $KB \cup \{C \sqsubseteq \bot\}$ satisfiable?	value of $dl_{unsatisfiable}(C)$
yes	–	`true`
no	yes	$\bot$
no	no	`false`

Querying for extensional disjointness of concepts is treated similarly, by reducing it to subsumption: two concepts C and D are disjoint iff $C \sqsubseteq \neg D$.

The query whether $C(a)$ holds can be resolved as in the following table. Note that $C(a)$ holds if it is true in *all* models.

result of the query: $C(a)$	result of the query: $\neg C(a)$	value of $dl_{member}(C, a)$
yes	–	`true`
no	yes	`false`
no	no	$\bot$

The query $dl_{has\_role}(I_1, R, I_2)$ provides information whether two individuals I_1 and I_2 are connected via a role R. When $\langle I_1, I_2 \rangle \in R$ then the DL-atom is `true`. To evaluate the other truth-values, we have to restrict ourselves to the known individuals, as it is not possible in OWL to ask for negated roles [15]. When querying whether two individuals are connected via a role, it is a sensible assumption that this might be possible, i.e. that $\langle I_1, X \rangle \in R$ or $\langle X, I_2 \rangle \in R$. So we assign the value `false` to queries when there exists either an $X \neq I_2$ with $\langle I_1, X \rangle \in R$ or an $Y \neq I_2$ with $\langle Y, I_2 \rangle \in R$ but $\langle I_1, I_2 \rangle \notin R$. All other pairs of individuals get the value $\bot$.

To integrate these DL-atoms flawlessly with the semantics of our logic programming environment which is based on fixed points, we need to guarantee that the values of the DL-atoms are monotone w.r.t. the knowledge order. For now, we assume that the knowledge-base is static, i.e. it cannot change during the program evaluation. Then the evaluation of the DL-atoms always yields the same result, and thus, trivially, is monotone.

The implemented system, called PrOWLog, is available for download from http://logic.aifb.uni-karlsruhe.de/wiki/PrOWLog.

5 An Example

We exemplify our approach by extending an example given in [1], formalising a judge's decision process, as given by the following rules.

$$\begin{aligned} \mathit{is\_suspect} &\leftarrow \mathit{has\_motive} \vee \mathit{has\_witness} \\ \mathit{is\_cleared} &\leftarrow \neg \mathit{contradict\_alibi} \wedge \mathit{has\_alibi} \\ \mathit{charge} &\leftarrow \mathit{is\_suspect} \oplus \neg \mathit{is\_cleared} \end{aligned}$$

The judge collects information suggesting that a person is suspect as well as information that indicates that the person is cleared. To support suspicion he collects information about the existence of a motive or a witness (first line). To enforce innocence the judge considers an alibi, but only if this is not contradicted by the defendant's testimony (second line). Finally he combines this information (third line). Assume now that the only information the judge has about some person is $\mathit{has\_witness} \leftarrow \texttt{false}$. Only relying on this, the suspect shouldn't be charged. Based on the closed-world assumption we get $\mathit{has\_motive} = \texttt{false}$ and thus $\mathit{is\_suspect} = \texttt{false}$. As $\mathit{has\_alibi} = \texttt{false}$, we obtain $\mathit{is\_cleared} = \texttt{false}$. So when evaluating *charge* the information is contradictory and *charge* gets the value $\top$.

Using the open-world assumption, giving all atoms the default value $\bot$, we get $\mathit{is\_suspect} = \bot$ because $\mathit{has\_motive} = \bot$ and $\texttt{false} \vee \bot = \bot$. Since we know nothing about *has_alibi* and *contradict_alibi*, the default assumption is used again and we get $\mathit{is\_cleared} = \bot$ and finally $\mathit{charge} = \bot$. So neither of the two established assumptions work in a satisfactory way. Consider now the mixed hypothesis H_m defined as follows: $H_m(\mathit{has\_witness}) = \texttt{false}$, $H_m(\mathit{has\_motive}) = \texttt{false}$, $H_m(\mathit{has\_alibi}) = \bot$, $H_m(\mathit{contradict\_alibi}) = \bot$. Then, like under the closed-world assumption, *is_suspect* is false. The contradiction we encountered, however, does not exist any more as $\mathit{is\_cleared} = \bot$ which reflects that the information is not sufficient to make a decision. Consequently $\mathit{charge} = \texttt{false}$. This illustrates that the first line of the program is devised according to the closed-world assumption. The second line however is based on a different idea: For *is_cleared* to become false, $\mathit{has\_alibi} = \texttt{false}$ is already sufficient. So the meaning of $\mathit{has\_alibi} = \texttt{false}$ is that it has been proven that nobody can provide an alibi for the defendant. Then we need also the possibility to model the fact, that just no alibi is known, which corresponds to $\mathit{has\_alibi} = \bot$, and which should be the default case. So the second line is conceived with an open-world setting in mind. H-founded models enable the use of such programs despite the different approaches involved.

To complete the example, it could be assumed that the judge draws his knowledge from an OWL DL knowledge base, by means of the following rules

which query a knowledge base about a person Ted who is under investigation.

$$has\_motive \leftarrow dl_{member}(dl\_has\_motive, Ted)$$
$$has\_witness \leftarrow dl_{member}(dl\_has\_witness, Ted)$$
$$has\_alibi \leftarrow dl_{member}(dl\_has\_alibi, Ted)$$
$$contradict\_alibi \leftarrow dl_{member}(dl\_contradict\_alibi, Ted)$$

By means of our hybrid semantics, the system will respond with the desired answer.

6 Related work

Our approach using DL-atoms to link rule-based and ontology-based reasoning is inspired by the approach of Eiter et. al. presented in [16], where *extended logic programs* and the *answer set semantics* [17] are modified to incorporate DL-atoms to query external reasoners. This approach permits the flow of information in both directions from the DL-enhanced program to the reasoner and vice versa. Extended logic programs make use of two negation operators, distinguishing explicitly negation as failure and classical negation. The any-world-semantics permits to manage this naturally, giving the negation operator of the lattice different meanings respective to the default assumption of the negated expression. In [18] Eiter et. al. generalize their approach to so called HEX-Programs, where the DL-atoms are replaced by atoms permitting to access a variety of different external sources, not only DL-reasoners. To accomplish this, the rule syntax and the answer set semantics are extended. The evaluation of these programs is made possible by a splitting algorithm based on the dependency structure in the program. Also based on DL-atoms, our approach was developed from another perspective. Given the elegant yet expressive any-world semantics and the ease of use of the hypotheses in KKS, we provide a logic programming environment with access to description logics, while remaining close to Prolog programming. We emphasize the use of the particular kind of information that can be drawn from DL-reasoners as an open-world based system with an intuitive semantics.

Motik and Rosati present in [19] an approach for a system combining rules and DL into one formalism. Based on MKNF [20] they join a decidable FOL fragment with logic programming rules. The modality operators in their so called *hybrid MKNF knowledge bases* allow to formulate rules to enforce closed world reasoning while maintaining the open world assumption for the DL-part. Their system also subsumes Rosati's approach in [21]. There and more detailed in [22] he discusses the relation of open and closed semantics in these hybrid systems.

7 Conclusions and Further Work

We have presented the hybrid reasoning system PrOWLog, which allows to combine OWL DL with Prolog in such a way that the open-world semantics of OWL

DL can be captured within the Prolog system. To the best of our knowledge, this is the first work which integrates a logic programming language and OWL in such a way.

We perceive basically two lines of further research to follow up on our results. On the one hand, studies remain to be done which show that the approach is useful in practice. We believe that in particular an integration with F-Logic reasoners is worth investigating, as F-Logic and OWL are two complementary ontology paradigms, which are both used in practice. On the other hand, it remains to be investigated whether the integration of Prolog and OWL can be strengthened by weakening the layering, i.e. by allowing some flow of information back to the OWL knowledge base, perhaps in a way similar to [18].

References

1. Loyer, Y., Straccia, U.: Any-world assumptions in logic programming. Theoretical Computer Science **342** (2005) 351–381
2. Kifer, M., Lausen, G., Wu, J.: Logical foundations of object-oriented and frame-based languages. Journal of the ACM **42** (1995) 741–843
3. Angele, J., Lausen, G.: Ontologies in F-logic. In Staab, S., Studer, R., eds.: Handbook on Ontologies. Springer (2004) 29–50
4. Ginsberg, M.L.: Multivalued logics: A uniform approach to inference in artificial intelligence. Computational Intelligence **4** (1988) 265–316
5. Belnap, N.D.: A useful four-valued logic. In Epstein, G., Dunn, J.M., eds.: Modern Uses of Multiple-Valued Logic. Reidel, Dordrecht, Netherlands (1977) 5–37
6. Fitting, M.: The family of stable models. Journal of Logic Programming **17** (1993) 197–225
7. Fitting, M.: A kripke-kleene semantics for logic programs. Journal of Logic Programming **2** (1985) 295–312
8. Fitting, M.C.: Bilattices in logic programming. In: 20th International Symposium on Multiple-Valued Logic, Charlotte. IEEE CS Press, Los Alamitos (1990) 238–247
9. van Gelder, A., Ross, K., Schlipf, J.S.: The well-founded semantics for general logic programs. Journal of the ACM **38** (1991) 620–650
10. Tarski, A.: A lattice-theoretic fixpoint theorem and its applications. Pacific Journal of Mathematics **5** (1955) 285–309
11. Motik, B.: Reasoning in description logics using resolution and deductive databases. PhD thesis, Universität Karlsruhe (2006)
12. Horrocks, I., Sattler, U., Tobies, S.: Practical reasoning for expressive description logics. In Ganzinger, H., McAllester, D., Voronkov, A., eds.: Proceedings of the 6th International Conference on Logic for Programming and Automated Reasoning (LPAR'99). Volume 1705 of Lecture Notes in Artificial Intelligence. Springer, Berlin/Heidelberg (1999) 161–180
13. Fitting, M.: Fixpoint semantics for logic programming – a survey. Theoretical Computer Science **278** (2002) 25–51
14. Kunen, K.: Negation in logic programming. Journal of Logic Programming **4** (1987) 289–308
15. Eiter, T., Lukasiewicz, T., Schindlauer, R., Tompits, H.: Well-founded semantics for description logic programs in the semantic web. In Antoniou, G., Boley, H., eds.: Rules and Rule Markup Languages for the Semantic Web: Third International

Workshop, RuleML 2004, Hiroshima, Japan, November 8, 2004. Proceedings. Volume 3323 of Lecture Notes in Computer Science. Springer, Berlin/Heidelberg (2004) 81–97
16. Eiter, T., Lukasiewicz, T., Schindlauer, R., Tompits, H.: Combining answer set programming with description logics for the semantic web. In Dubois, D., Welty, C.A., Williams, M.A., eds.: KR2004: Principles of Knowledge Representation and Reasoning. AAAI Press, Menlo Park, California (2004) 141–151
17. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Computing **9** (1991) 365–386
18. Eiter, T., Ianni, G., Schindlauer, R., Tompits, H.: Effective integration of declarative rules with external evaluations for semantic-web reasoning. In Sure, Y., Domingue, J., eds.: The Semantic Web: Research and Applications, 3rd European Semantic Web Conference, ESWC 2006, Proceedings. Volume 4011 of Lecture Notes in Computer Science. Springer, Berlin/Heidelberg (2006) 273–287
19. Motik, B., Rosati, R.: Closing semantic web ontologies. Technical report, University of Manchester, UK (2006)
20. Lifschitz, V.: Nonmonotonic databases and epistemic queries. In: Proceedings of IJCAI-91, San Mateo, CA., Morgan Kaufmann (1991) 381–386
21. Rosati, R.: DL+log: Tight integration of description logics and disjunctive datalog. In Doherty, P., Mylopoulos, J., Welty, C.A., eds.: Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning, Lake District of the United Kingdom, June 2-5, 2006, AAAI Press (2006) 68–78
22. Rosati, R.: Semantic and computational advantages of the safe integration of ontologies and rules. In Fages, F., Soliman, S., eds.: Principles and Practice of Semantic Web Reasoning, Third International Workshop, PPSWR 2005, Dagstuhl Castle, Germany, September 11-16, 2005, Proceedings. Volume 3703 of Lecture Notes in Computer Science., Springer (2005) 50–64