Implementing Query Answering for Hybrid MKNF Knowledge Bases

Ana Sofia Gomes, José Júlio Alferes and Terrance Swift

CENTRIA, Departamento de Informática
Faculdade Ciências e Tecnologias
Universidade Nova de Lisboa
2829-516 Caparica, Portugal

Abstract. Ontologies and rules are usually loosely coupled in knowledge representation formalisms. In fact, ontologies use open-world reasoning while the leading semantics for rules use non-monotonic, closed-world reasoning. One exception is the tightly-coupled framework of Minimal Knowledge and Negation as Failure (MKNF), which allows statements about individuals to be jointly derived via entailment from an ontology and inferences from rules. Nonetheless, the practical usefulness of MKNF has not always been clear, although recent work has formalized a general resolution-based method for querying MKNF when rules are taken to have the well-founded semantics, and the ontology is modeled by a general Oracle. That work leaves open what algorithms should be used to relate the entailments of the ontology and the inferences of rules. In this paper we provide such algorithms, and describe the implementation of a query-driven system, CDF-Rules, for hybrid knowledge bases combining both (non-monotonic) rules under the well-founded semantics and a (monotonic) ontology, represented by a CDF ($\mathcal{ALCQ}$) theory.

1 Introduction

Ontologies and rules offer distinctive strengths for the representation and transmission of knowledge over the Semantic Web. Ontologies offer the deductive advantages of first-order logics with an open domain while guaranteeing decidability. Rules employ non-monotonic (closed-world) reasoning that can formalize scenarios under locally incomplete knowledge; rules also offer the ability to reason about fixed points (e.g. reachability) which cannot be expressed within first-order logic.

Several factors influence the decision of how to combine rules and ontologies into a hybrid knowledge base. The choice of semantics for the rules, such as the answer-set semantics [5] or the well-founded semantics (WFS) [14], can greatly affect the behavior of the knowledge base system. The answer set semantics offers several advantages: for instance, description logics can be translated into the answer set semantics providing a solid basis for combining the two paradigms [9,11]. WFS is weaker than the answer-set semantics (in the sense that it is more skeptical), having the advantage of a lower complexity, and that it can be evaluated in a query-oriented, Prolog-like, fashion and having, in fact, been integrated in Prolog systems.

Several formalisms have concerned themselves with combining ontologies with WFS rules [3,4,7]. Among these, the Well-Founded Semantics for Hybrid MKNF knowledge bases (*MKNF*$_{WFS}$), introduced in [7] and overviewed in Section 2 below, is the only one which allows knowledge about instances to be fully inter-definable between rules and an ontology that is taken as a parameter of the formalism. *MKNF*$_{WFS}$ assigns a well founded semantics to Hybrid MKNF knowledge bases, is sound w.r.t. the original semantics of [10] and, as in [10], allows the knowledge base to have both closed- and open-world (classical) negation.

Example 1. The following fragment, adapted from an example [10], concerning car insurance premiums illustrates several properties of *MKNF*$_{WFS}$. The ontology consists of the axioms:

$$nonMarried \equiv \neg married \quad \neg married \sqsubseteq highRisk \quad \exists Spouse.T \sqsubseteq married$$

while the rule base consists of the rules:

$$\mathbf{K}\, nonMarried(X) \leftarrow \mathbf{K}\, person(X), \mathbf{not}\; married(X).$$
$$\mathbf{K}\, discount(X) \leftarrow \mathbf{not}\; spouse(X,Y), \mathbf{K}\, person(X), \mathbf{K}\, person(Y).$$
$$\mathbf{K}\, surcharge(X) \leftarrow \mathbf{K}\, highRisk(X), \mathbf{K}\, person(X).$$

Note that *married* and *nonMarried* are defined both by axioms in the ontology and by rules. Within the rule bodies, literals with the **K** or **not** operators (e.g. $\mathbf{K}\, highRisk(X)$) may require information both from the ontology and from other rules; other literals are proven directly by the other rules (e.g. *person(X)*).

Suppose *person(john)* were added as a fact (in the rule base). Under closed-world negation, the first rule would derive *nonMarried(john)*. By the first ontology axiom, $\neg$ *married(john)* would hold, and by the second axiom *highRisk(john)* would also hold. By the third rule, $surcharge(john)$ would also hold. Thus the proof of $surcharge(john)$ involves interdependencies between the rules with closed-world negation, and the ontology with open-world negation. At the same time the proof of $surcharge(john)$ is *relevant* in the sense that properties of other individuals do not need to be considered.

In the original definition of *MKNF*$_{WFS}$, the inter-dependencies of the ontology and rules were captured by a bottom-up fixed-point operator with multiple levels of iterations. Recently, a query-based approach to hybrid MKNF knowledge bases, called SLG($\mathcal{O}$), has been developed using tabled resolution [1]. SLG($\mathcal{O}$) is sound and complete, as well as terminating for various classes of programs (e.g. datalog). In addition SLG($\mathcal{O}$) is relevant in the sense of Example 1. This relevancy is a critical requirement for scalability in real domains application (e.g. in the area of Semantic Web): without relevance a query about a particular individual I may need to derive information about other individuals even if those individuals were not connected with I through rules or axioms. SLG($\mathcal{O}$) serves as a theoretical framework for query evaluation of *MKNF*$_{WFS}$ knowledge bases, but it models the inference mechanisms of an ontology abstractly, as an oracle. While this abstraction allows the resolution method to be parameterized by different ontology formalisms in the same manner as *MKNF*$_{WFS}$, it leaves open details of how the ontology and rules should interact and these details must be accounted for in an implementation.

This paper describes, in Section 4, the design and implementation of a working prototype query evaluator[1] for *MKNF*$_{WFS}$, called *CDF-Rules*, which fixes the ontology part to $\mathcal{ALCQ}$ theories, and makes use of the prover from XSB's ontology management, the Coherent Description Framework (CDF) [12] (overviewed in Section 3). To the best of our knowledge, this implementation is the first working query-driven implementation for Hybrid MKNF knowledge bases, combining rules and ontology and complete w.r.t. the well-founded semantics.

2 MKNF Well-Founded Semantics

Hybrid MKNF knowledge bases as introduced in [10] are essentially formulas in the logics of minimal knowledge and negation as failure (MKNF) [8], i.e. first-order logics with equality and two modal operators $\mathbf{K}$ and $\mathbf{not}$, which allow inspection of the knowledge base. Intuitively, given a first-order formula φ, $\mathbf{K}\varphi$ asks whether φ is known while $\mathbf{not}\varphi$ is used to check whether φ is not known. A Hybrid MKNF knowledge base consists of two components, a decidable description logic (DL) knowledge base, translatable into first-order logic, and a finite set of rules of modal atoms.

Definition 1. *Let $\mathcal{O}$ be a DL knowledge base built over a language $\mathcal{L}$ with distinguished sets of countably infinitely many variables N_V, along with finitely many individuals N_I and predicates (also concepts) N_C. An atom $P(t_1, \ldots, t_n)$ where $P \in N_C$ and $t_i \in N_V \cup N_I$ is called a* DL-atom *if P occurs in $\mathcal{O}$, otherwise it is called* non-DL-atom. *An MKNF rule r has the following form where H_i, A_i, and B_i are atoms: (1) $\mathbf{K}H \leftarrow \mathbf{K}A_1, \ldots, \mathbf{K}A_n, \mathbf{not}B_1, \ldots, \mathbf{not}B_m$. H is called the* (rule) head *and the sets $\{\mathbf{K}A_i\}$, and $\{\mathbf{not}B_j\}$ form the* (rule) body. *Atoms of the form $\mathbf{K}A$ are also called* positive literals *or* modal $\mathbf{K}$-atoms *while atoms of the form $\mathbf{not}A$ are called* negative literals *or* modal $\mathbf{not}$-atoms. *A rule r is* positive *if $m = 0$ and a* fact *if $n = m = 0$. A* program *$\mathcal{P}$ is a finite set of MKNF rules and a* hybrid MKNF knowledge base *$\mathcal{K}$ is a pair $(\mathcal{O}, \mathcal{P})$.*

For decidability DL-safety is applied which basically constrains the use of rules to individuals actually appearing in the knowledge base under consideration. Formally, an MKNF rule r is *DL-safe* if every variable in r occurs in at least one non-DL-atom $\mathbf{K}B$ occurring in the body of r. A hybrid MKNF knowledge base $\mathcal{K}$ is *DL-safe* if all its rules are DL-safe (for more details we refer to [10]).

The well-founded MKNF semantics as presented in [7] is based on a complete three-valued extension of the original MKNF semantics. However, here, as we are only interested in querying for literals and conjunctions of literals, we limit ourselves to the computation of what is called the well-founded partition in [7]: basically the atoms which are true and false. For that reason, and in correspondence to logic programming, we will name this partition the well-founded model. At first, we recall some notions from [7] which will be useful in the definition of the operators for obtaining that well-founded model.

[1] The implementation is freely available from the XSB CVS repository.

Definition 2. *Consider a hybrid MKNF knowledge base $\mathcal{K} = (\mathcal{O}, \mathcal{P})$. The* set of $\mathbf{K}$-atoms of $\mathcal{K}$, *written* $\mathsf{KA}(\mathcal{K})$, *is the smallest set that contains (i) all modal* $\mathbf{K}$*-atoms occurring in* $\mathcal{P}$*, and (ii) a modal atom* $\mathbf{K}\xi$ *for each modal atom* $\mathbf{not}\xi$ *occurring in* $\mathcal{K}$*. Furthermore, for a set of modal atoms* S*,* S_{DL} *is the subset of DL-atoms of* S*, and* $\widehat{S} = \{\xi \mid \mathbf{K}\xi \in S\}$.

Basically all modal atoms appearing in the rules are collected in $\mathsf{KA}(\mathcal{K})$. The other notions are useful below when defining an operator on hybrid MKNF KB's.

To guarantee that all atoms that are false in the ontology are also false by default in the rules, we introduce new positive DL atoms which represent first-order false DL atoms, and a program transformation making these new modal atoms available for reasoning in the respective rules.

Definition 3. *Let* $\mathcal{K}$ *be a DL-safe hybrid MKNF knowledge base. We obtain* $\mathcal{K}^+$ *from* $\mathcal{K}$ *by adding an axiom* $\neg P \sqsubseteq$ *NP for every DL atom* P *which occurs as head in at least one rule in* $\mathcal{K}$ *where NP is a new predicate not already occurring in* $\mathcal{K}$*. Moreover, we obtain* $\mathcal{K}^*$ *from* $\mathcal{K}^+$ *by adding* $\mathbf{not}$ $NP(t_1, \ldots, t_n)$ *to the body of each rule with a DL atom* $P(t_1, \ldots, t_n)$ *in the head.*

By $\mathcal{K}^+$, *NP* represents $\neg P$ (with its corresponding arguments) and $\mathcal{K}^*$ introduces a restriction on each rule with such a DL atom in the head saying intuitively that the rule can only be used to conclude the head if the negation of its head cannot be proved[2].

We continue now by recalling the definition in [7] of an operator $T_{\mathcal{K}}$ which allows conclusions to be drawn from positive hybrid MKNF knowledge bases.

Definition 4. *For* $\mathcal{K}$ *a positive DL-safe hybrid MKNF knowledge base,* $R_{\mathcal{K}}$*,* $D_{\mathcal{K}}$*, and* $T_{\mathcal{K}}$ *are defined on the subsets of* $\mathsf{KA}(\mathcal{K}^*)$ *as follows:*

$$\begin{aligned}
R_{\mathcal{K}}(S) &= S \cup \{\mathbf{K}H \mid \mathcal{K} \textit{ contains a rule of the form (1) such that } \mathbf{K}A_i \in S \\
&\qquad \textit{for each } 1 \leq i \leq n\} \\
D_{\mathcal{K}}(S) &= \{\mathbf{K}\xi \mid \mathbf{K}\xi \in \mathsf{KA}(\mathcal{K}^*) \textit{ and } \mathcal{O} \cup \widehat{S}_{DL} \models \xi\} \cup \{\mathbf{K}Q(b_1, \ldots, b_n) \mid \\
&\qquad \mathbf{K}Q(a_1, \ldots, a_n) \in S \setminus S_{DL}, \mathbf{K}Q(b_1, \ldots, b_n) \in \mathsf{KA}(\mathcal{K}^*), \textit{ and} \\
&\qquad \mathcal{O} \cup \widehat{S}_{DL} \models a_i \approx b_i \textit{ for } 1 \leq i \leq n\} \\
T_{\mathcal{K}}(S) &= R_{\mathcal{K}}(S) \cup D_{\mathcal{K}}(S)
\end{aligned}$$

$R_{\mathcal{K}}$ derives consequences from the rules while $D_{\mathcal{K}}$ obtains knowledge from the ontology $\mathcal{O}$, both from non-DL-atoms and the equalities occurring in $\mathcal{O}$. The $\approx$ operator defines a congruence relation between individuals.

The operator $T_{\mathcal{K}}$ is shown to be monotonic in [7] so, by the Knaster-Tarski theorem, it has a unique least fixpoint, denoted $\mathsf{lfp}(T_{\mathcal{K}})$, which is reached after a finite number of iteration steps.

The computation follows the alternating fixpoint construction [13] of the well-founded semantics for logic programs which necessitates turning a hybrid MKNF knowledge base into a positive one to make $T_{\mathcal{K}}$ applicable.

[2] Note that $\mathcal{K}^+$ and $\mathcal{K}^*$ are still hybrid MKNF knowledge bases, so we only refer to $\mathcal{K}^+$ and $\mathcal{K}^*$ explicitly when it is necessary.

Definition 5. *Let $\mathcal{K}_G = (\mathcal{O}, \mathcal{P}_G)$ be a ground DL-safe hybrid MKNF knowledge base and let $S \subseteq \mathsf{KA}(\mathcal{K}_G)$. The* MKNF transform *$\mathcal{K}_G/S = (\mathcal{O}, \mathcal{P}_G/S)$ is obtained by $\mathcal{P}_G/S$ containing all rules $H \leftarrow A_1, \ldots, A_n$ for which there exists a rule $\mathbf{K}H \leftarrow \mathbf{K}A_1, \ldots, \mathbf{K}A_n, \mathbf{not}B_1, \ldots, \mathbf{not}B_m$ in $\mathcal{P}_G$ with $\mathbf{K}B_j \notin S$ for all $1 \leq j \leq m$.*

This resembles the transformation known from answer-sets [5] of logic programs and the following two operators are defined.

Definition 6. *Let $\mathcal{K} = (\mathcal{O}, \mathcal{P})$ be a nondisjunctive DL-safe hybrid MKNF knowledge base and $S \subseteq \mathsf{KA}(\mathcal{K}^*)$. We define: $\Gamma_{\mathcal{K}}(S) = \mathsf{lfp}(T_{\mathcal{K}^+_G/S})$, and $\Gamma'_{\mathcal{K}}(S) = \mathsf{lfp}(T_{\mathcal{K}^*_G/S})$.*

Both operators are shown to be antitonic [7], hence their composition is monotonic and form the basis for defining the well-founded MKNF model. Here we present its alternating computation.

$$\begin{aligned} \mathbf{T}_0 &= \emptyset & \mathbf{TU}_0 &= \mathsf{KA}(\mathcal{K}^*) \\ \mathbf{T}_{n+1} &= \Gamma_{\mathcal{K}}(\mathbf{TU}_n) & \mathbf{TU}_{n+1} &= \Gamma'_{\mathcal{K}}(\mathbf{T}_n) \\ \mathbf{T}_\omega &= \bigcup \mathbf{T}_n & \mathbf{TU}_\omega &= \bigcap \mathbf{TU}_n \end{aligned}$$

Note that by finiteness of the ground knowledge base the iteration stops before reaching ω. It was shown in [7] that the sequences are monotonically increasing, decreasing respectively, and that $\mathbf{T}_\omega$ and $\mathbf{TU}_\omega$ form the well-founded model:

Definition 7. *Let $\mathcal{K} = (\mathcal{O}, \mathcal{P})$ be a DL-safe hybrid MKNF knowledge base and let $\mathbf{T}_{\mathcal{K}}, \mathbf{TU}_{\mathcal{K}} \subseteq \mathsf{KA}(\mathcal{K})$ with $\mathbf{T}_{\mathcal{K}}$ being $\mathbf{T}_\omega$ and $\mathbf{TU}_{\mathcal{K}}$ being $\mathbf{TU}_\omega$, both restricted to the modal atoms only occurring in $\mathsf{KA}(\mathcal{K})$. Then $M_{WF} = \{\mathbf{K}A \mid A \in \mathbf{T}_{\mathcal{K}}\} \cup \{\mathbf{K}\pi(\mathcal{O})\} \cup \{\mathbf{not}A \mid A \in \mathsf{KA}(\mathcal{K}) \setminus \mathbf{TU}_{\mathcal{K}}\}$ is the* well-founded MKNF model *of $\mathcal{K}$, where $\pi(\mathcal{O})$ denotes the first order logic formula equivalent to the ontology $\mathcal{O}$ (for detail on the translation of $\mathcal{O}$ into first order logic see [10]) .*

All modal $\mathbf{K}$-atoms in M_{WF} are true, all modal $\mathbf{not}$-atoms are false and all other modal atoms from $\mathsf{KA}(\mathcal{K})$ are undefined.

As shown in [7], the well founded model is sound with respect to the original semantics of [10], i.e. all atoms true (resp. false) in the well founded model are also true (resp. false) according to [10]. In fact, the relation between the semantics of [7] and [10], is tantamount to that of the well founded semantics and the answer-sets semantics of logic programs.

3 XSB Prolog and the Coherent Description Framework

Our implementation makes use of XSB Prolog (`xsb.sourceforge.net`) to implement *MKNF*$_{WFS}$ for two reasons. First, XSB's tabling engine evaluates rules according to WFS, and ensures rule termination for programs with the *bounded term-size property*. Second, the implementation directly uses the prover from XSB's ontology management, the Coherent Description Framework (CDF) [12].

CDF has been used in numerous commercial projects, and was originally developed as a proprietary tool by the company XSB, Inc although significant portions of it have been made open source, and are available in the standard XSB package release. Over the last 6 years CDF has been used to support extraction of information about aircraft parts from free-text data fields, about medical supplies and electronic parts from websites and electronic catalogs, and about the specifics of mechanical parts from scanned technical drawings. Also, CDF is used to maintain models of graphical user interfaces that are driven by XSB and its graphics package, XJ. Next, we discuss a few features of CDF that are relevant to the implementation described in Section 4.

Commercial use has driven CDF to support efficient query answering from Prolog. As a result, ontologies in CDF can have a restricted, tractable form. *Type-0* ontologies do not allow representation of negation or disjunction within the ontology itself, and implicitly use the closed-world assumption. As such, Type-0 ontologies resemble a frame-based representation more than a description logic, and do not add any complexity to query evaluation beyond that of WFS. *Type-1* ontologies use open-world negation and support $\mathcal{ALCQ}$ description logics. The vast majority of knowledge used by XSB, Inc. is maintained in Type-0 ontologies; Type 1 ontologies are used for small projects in XSB, Inc. and for research.

Regardless of the type of the ontology, primitive classes in CDF are represented by terms *cid(Identifier, Namespace)*, instances by terms *oid(Identifier, Namespace)*, and relations by terms *rid(Identifier, Namespace)*. The atom $isa/2$ is used to state inclusion: whether the inclusion is a subclass, element of, or subrelation depends on the type of the term, and not all combinations of types of terms are allowed in a CDF program. Relational atoms in CDF have the form $hasAttr(Term_1, Rel_1, Term_2)$ which has the meaning $Term_1 \sqsubseteq \exists Rel_1.Term_2$; $allAttr(Term_1, Rel_1, Term_2)$ which has the meaning $Term_1 \sqsubseteq \forall Rel_1.Term_2$, along with other forms that designate cardinality constraints on relations. Query answering to Type-0 ontologies is supported by tabling to implement inheritance and by tabled negation so that only the most specific answers to a query are returned to a user.

Unlike Type-0, Type-1 ontologies also allow atoms $necessCond(Term_1, CE)$ where CE can be any $\mathcal{ALCQ}$ class expression over CDF terms. Because they use open-world negation, atoms for Type-1 ontologies cannot be directly queried; rather they are queried through goals such as $allModelsEntails(Term, ClassExpr)$, succeeding if $Term \sqsubseteq ClassExpr$ is provable in the current state of the ontology. Type-1 ontologies deduce entailment using a tableau prover written in Prolog.

Regardless of the type of the ontology, atoms such as $isa/2$, $hasAttr/2$, etc. can be defined extensionaly via Prolog facts, or intensionaly via Prolog rules. Intensional definitions are used in Type-0 database so that atoms can be lazily defined by querying a database or analyzing a graphical model: their semantics is outside that of CDF. At the same time, intensional definitions in a Type-1 ontology provides a basis for the tableau prover to call rules, as is required to support the interdependencies of *MKNF*$_{WFS}$.

4 Goal-Driven MKNF Implementation

In this section we describe the algorithms and the design of a goal driven implementation for Hybrid MKNF Knowledge Bases under the Well Founded Semantics. Our solution makes use of XSB's SLG Resolution [2] for the evaluation of a query, together with tableaux mechanisms supported by CDF theorem prover to check entailment on the ontology. In this section we assume a general knowledge of tabled logic programs.

4.1 A Query-Driven Iterative Fixed Point

At an intuitive level, a query to *CDF-Rules* is evaluated in a relevant (top-down like) manner with tabling, through SLG resolution [2], until the selected goal is a literal l formed over a DL-atom. At that point, in addition to further resolution, the ontology also uses tableau mechanisms to derive l. However, as a tableau proof of l may require propositions (literals) inferred by other rules, considerable care must be taken to integrate the tableau proving with rule-based query evaluation.

In its essence, a tableau algorithm decides the entailment of a formula f w.r.t. an ontology $\mathcal{O}$ by trying to construct a common *model* for $\neg f$ and $\mathcal{O}$, sometimes called a *completion graph*. If such a model can not be constructed, $\mathcal{O} \models f$; otherwise $\mathcal{O}$ does not entail f. Similar to other description logic provers, the CDF theorem prover attempts to traverse as little of an ontology as possible when proving f. As a result, when the prover is invoked on an atom A, the prover attempts to build a model for the underlying individual(s) to which A refers, and explores other individuals only as necessary.

Now, given the particular interdependence between the rules and the ontology in *MKNF*$_{WFS}$, the prover must consider the knowledge inferred by the rules in the program for the entailment proof, as a DL-atom can be derived by rules, which in turn may rely on other DL-atoms proven by the ontology. Thus, for a query to a DL-atom $p(o)$, the idea is to iteratively compute a model for o, deriving at each iteration new information about the roles and classes of o, along with information about other individuals related to o either in the ontology (via CDF's tableau algorithm) or in the rules (via SLG procedures) until a fixed point is reached.

We start by illustrating the special case of positive knowledge bases without default negation in the rules.

Example 2. Consider the following KB (with the program on the left and the ontology on the right[3]) and the query $third(X)$:

$$\begin{array}{ll} \mathbf{K}\, third(X) \leftarrow p(X), \mathbf{K}\, second(X). & \\ \mathbf{K}\, first(callback). & First \sqsubseteq Second \\ p(callback). & \end{array}$$

The query resolves against the rule for $third(X)$, leading to the goals $p(X)$ and $second(X)$. The predicate p, although not a DL-atom, assures DL-safety, restricting the application of the rules to known individuals. The call $p(X)$ returns true for $X = callback$. However, now the call $third(callback)$ (since X was bound to $callback$ by p) depends on the DL-atom $second(callback)$, corresponding in the ontology to the proposition $Second$. So the computation calls the CDF theorem prover which starts to derive a model for all the properties of the individual $callback$. Yet, in this computation, the proposition $Second$ itself depends on a predicate defined in the rules –

[3] To simplify reading we use the usual notation for the ontology, where the argument variable of a unary predicate is not displayed, and the first letter of the predicate's name is capitalized. For rules we use the usual logic programming notation, and omit the $\mathbf{K}$ before non-DL atoms. In fact, in the implementation the ontology must be written according to CDF syntax, and in the rules the modal operators $\mathbf{K}$ and $\mathbf{not}$ are replaced by (meta-)predicates $known/1$ and $dlnot/1$, respectively (see Section 4.2).

$First$. It is intuitive that the evaluation of the query $third(callback)$ must be done iteratively – the (instantiated) goal $third(callback)$ should suspend (using tabling) until $second(callback)$ is resolved. Furthermore, $second(callback)$ needs first to prove $first(callback)$ from the rules. In general, goals to DL-atoms may need to suspend in order to compute an iterative fixed point, after which they may either succeed or fail.

We formalize the actions in Example 2 on the special case of definite programs as follows.

Definition 8. *Let $\mathcal{K} = (\mathcal{O}, \mathcal{P})$ be a DL-safe hybrid MKNF knowledge base, where $\mathcal{P}$ does not contain default negation. Let $\mathcal{I}$ be a fixed set of individuals. The function Tableaux$(\mathcal{O})$ computes for a theory $\mathcal{O}$ the entailments of $\mathcal{O}$ for $\mathcal{I}$, disregarding the rules component. The function $SLG(\mathcal{P})$ computes via tabling the set of DL-atoms true in the minimal model of $\mathcal{P}$ for a set of individuals, $\mathcal{I}$, disregarding the ontology component. The model is obtained as the least fixed point of the alternative sequence:*

$$\begin{array}{ll}
D_0 = Tableaux(\mathcal{O}) & R_0 = SLG(\mathcal{P}) \\
D_1 = Tableaux(\mathcal{O} \cup R_0) & R_1 = SLG(\mathcal{P} \cup D_0) \\
\ldots & \ldots \\
D_n = Tableaux(\mathcal{O} \cup R_{n-1}) & R_n = SLG(\mathcal{P} \cup D_{n-1})
\end{array}$$

where n is odd and ≥ 2. The iteration stops when a fixed point in R_n is reached.

Definition 8 resembles Definition 4 of the operator $T_{\mathcal{K}}$ in Section 2. As in Definition 4, since it considers only positive rules, the operators SLG and $Tableaux$ are monotonic and thus a least fixed point is guaranteed to exist. Furthermore, the program respects DL-safety, which means that MKNF rules are lazily grounded with respect to the set of individuals (constants). Thus the program is finite and the fixed point can be obtained in a finite number of steps.

Definition 8 captures certain aspects of how the rules and ontology use each other as a way to derive new knowledge in *CDF-Rules*, via an alternating computation. However it does not capture cases in which the relevant set of individuals changes, or the presence of default negation in rule bodies. With regard to relevant individuals, since it is possible to define n-ary predicates in rules along with roles in the ontology, the query may depend on a set of several individuals. Therefore, the fixed point computation must take into account the entire set of individuals that the query depends on. This is done by tabling information about each individual in the set of individuals relevant to the query. This set may increase throughout the fixed point iteration as new dependency relations between individuals (including equality) are discovered. The iteration stops when it is not possible to derive anything more about these individuals, i.e., when all individuals in the set have reached a fixed point.

Example 3. Regarding default negation, consider the following knowledge base:

$$\begin{array}{lll}
\mathbf{K}\, third(X) \leftarrow p(X), \mathbf{K}\, second(X). & \mathbf{K}\, first(callback). & First \sqsubseteq Second \\
\mathbf{K}\, fourth(X) \leftarrow p(X), \mathbf{not}\, third(X). & p(callback). & Fourth \sqsubseteq Fifth
\end{array}$$

In this example a predicate $fourth(X)$ is defined at the expense of the negation of $third(X)$. Since $fourth(X)$ is defined in the rules, the negation is *closed world*, that is, $fourth(X)$ should only succeed if it is not possible to prove $third(X)$. Consequently, if we employed SLG resolution blindly, an iteration where the truth of $second(callback)$ had not been made available to the rules from the ontology might mistakenly fail the derivation of $third(callback)$ and so succeed $fourth(callback)$. Likewise, the rules may pass to the ontology knowledge, that after some iterations, no longer applies – in this case if the ontology were told that $fourth(callback)$ was true, it would mistakenly derive $Fifth$.

Example 3 shows a need to treat default negation carefully, as it requires re-evaluation when new knowledge is inferred. Recall how in Definition 6, operators $\Gamma_{\mathcal{K}}$ and $\Gamma'_{\mathcal{K}}$ are defined in order to address the problem of closed-world negation. Roughly, one step in $\Gamma_{\mathcal{K}}$ (or $\Gamma'_{\mathcal{K}}$) is defined as the application of $T_{\mathcal{K}}$ until reaching a fixed point. Applying $\Gamma'_{\mathcal{K}}$ followed by $\Gamma_{\mathcal{K}}$ is a monotonic operation and thus is guaranteed to have a least fixed point. In each dual application of $\Gamma_{\mathcal{K}}$ and $\Gamma'_{\mathcal{K}}$ two different models follow – a monotonically increasing model of trues (i.e. true predicates and propositions), and a monotonically decreasing model of trues and undefineds.

In a similar way, the implementation of ***CDF-Rules*** makes use of two fixed points: an *inner* fixed point where we apply Definition 8 corresponding to $T_{\mathcal{K}}$; and an *outer* fixed point for the evaluation of nots, corresponding to $\Gamma_{\mathcal{K}}$ (and $\Gamma'_{\mathcal{K}}$). In the outer operation, the evaluation of closed-world negation is made by a reference to the previous model obtained by $\Gamma_{\mathcal{K}}$. Thus in ***CDF-Rules***, $\mathbf{not}(A)$ succeeds if, in the previous *outer* iteration, A was not proven.

Example 4. As an illustration of the need for the application of the two fixed points, consider the knowledge base below and the query $c(X)$:

$$\mathbf{K}\, c(X) \leftarrow p(X), \mathbf{K}\, a(X), \mathbf{not}\, b(X) \qquad p(object). \qquad \mathbf{K}\, a(object). \qquad A \sqsubseteq B$$

When evaluating the query $c(X)$, X is first bound to $object$ by p, and then the iteration process of Definition 8 begins. Note that Definition 8 refers only to definite programs. To treat a rule like that for $c(X)$ as positive, each negative body literal is evaluated according to its value in the previous outer fixed point, or is simply evaluated as true in the first outer iteration. As will be seen, this is done lazily by ***CDF-Rules***. Accordingly, the rules infer $a(object)$, $p(object)$ and $c(object)$ for R_0. However in the first inner iteration the set of ontological entailments, D_0, is empty since $\mathcal{O} \not\models A$. In the second inner step the rules achieve the same fixed point as in the first, so $R_1 = R_0$, but the ontology derives B for $object$ in D_1. After sharing this knowledge, there is nothing more to infer by either components, and we achieve the first inner fixed point with:

$$T_1 = \{a(object), b(object), c(object), p(object)\}$$

So now, the second outer iteration will start the computation of the inner iteration again and, in this iteration, $\mathbf{not}$s are evaluated with respect to T_1. As a consequence, $c(object)$ fails, since $b(object) \in T_1$. The fixed point of the second inner iteration contains $p(object)$, $a(object)$ and $b(object)$, which is in fact the correct model for the object $object$. Afterwards, the outer iteration needs one more computational step to determine that a fixed point has been reached, and returns the model described. Since $c(object)$ is in the model, the query $c(X)$ succeeds for $X = object$.

The procedure for a lazily invoked iterative fixed point described above is summarized in Figure 1 using predicates that are described in detail in Section 4.2. The tabled predicate $known/3$ is used in each inner iteration to derive knowledge from the rules component, while $allModelsEntails/3$ infers knowledge from the ontology via a tableau proof. Within rules evaluated by $known/3$, the default negation of a DL-atom A is obtained by the predicate $dlnot(A)$, which succeeds if A was not proven in the last outer iteration. Whenever a role is encountered for an individual, a check is made to determine whether the related individual is already in the list of individuals in the fixed point, and the individual is added if not. The predicates $definedClass/2$ and $definedRole/3$ are used to obtain the relevant classes and roles defined for a given individual over a DL-safe MKNF Hybrid Knowledge Base. We assume that these predicates are defined explicitly by the compiler or programmer, but they can also be inferred via the DL-safe restriction. In fact, by bounding our program to DL-Safe rules, every rule in the hybrid knowledge base must contain a positive predicate that is only defined in the rules. This predicate limits the evaluation of the rules to *known* individuals, so that *CDF-Rules* can infer the set of individuals that are applicable to a given rule, that is, its *domain*.

The algorithm shown in Figure 1 creates two different sets corresponding to the application of the operator Γ of the *MKNF*$_{WFS}$ [7]. A credulous set, containing the atoms that are true or undefined; and a skeptical set of the atoms that are true (cf. Definition 7). As in the application of Γ, the T set is monotonically increasing, while TU set is monotonically decreasing. Finally, after computing the sets and achieving the fixed point, our algorithm returns the evaluation of $known(Query, Iteration - 1)$, where $Iteration$ represents the iteration where the outer fixed point was accomplished. Since the first outer set obtained corresponds to the first iteration in the TU set, this outer fixed point will be obtained in a TU iteration. Thus to check if $Query$ is true, we need to check if it is contained in the set inferred in $Iteration - 1$. If this is not the case, $Query$ is evaluated as undefined if it derived in $Iteration$, and as false otherwise.

4.2 Implementing *MKNF*$_{WFS}$ Components

We now provide a description of the various predicates in the algorithm of Figure 1, discuss the manner in which the rule and ontology components exchange knowledge, and how the fixed point is checked.

Rules Component As mentioned, inferences from rules are obtained using the predicate $known/1$ corresponding to $\mathbf{K}$ and $dlnot/1$ corresponding to $\mathbf{not}$. Cf. Figure 2, the call $known(A)$ with $A = p(O)$ first calls $computeFixedPoint(p(O))$ which begins the fixed point computation for O. $computeFixedPoint/1$ was summarized in Figure 1 and calls the lower-level $known/3$ and $dlnot/3$. Once the fixed point has been reached, the final iteration indices for O are obtained from a global store using *get_object_iter(p(O),Outer,Inner)*, and $known/3$ will be called again to determine whether $p(O)$ is true. This post-fixed point call to $known/3$ will simply check the table, and so will not be computationally expensive. $known/3$ is always called with the

```
Input: A query $Query$ to a DL-Atom
Output: Value of the input query in $MKNF_{WFS}$
addIndividuals($Query$,IndividualList);
foreach Individual in IndividualList do
    OutIter, InIter = 0;
    $S = S_1 = \{\}$;
    $P = P_1 = \{\}$;
    repeat
        $P = P_1$;
        repeat
            $S = S_1$;
            foreach Class in definedClass(Individual,Class) do
                Term = Class(Individual);
                $S_1 = S_1 \cup$ known(Term, OutIter, InIter);
                $S_1 = S_1 \cup$ allModelsEntails(Term, OutIter, InIter);
                $S_1 = S_1 \cup$ allModelsEntails(not Term, OutIter, InIter);
            end
            foreach Role in definedRole(Individual,Individual1,Role) do
                Term = Role(Individual,Individual1);
                add $Individual1$ to IndividualList if necessary
                $S_1 = S_1 \cup$ known(Term, OutIter, InIter);
                $S_1 = S_1 \cup$ allModelsEntails(Term, OutIter, InIter);
                $S_1 = S_1 \cup$ allModelsEntails(not Term, OutIter, InIter);
            end
            InIter++;
        until $S = S_1$ ;
        $P = S$;
        OutIter++;
    until $P = P_1$ ;
end
if known ($Query$,Final-1,Final) then
    return true
else
    if known($Query$,Final,Final) then
        return undefined
    else
        return false
    end
end
```

Fig. 1. The Top-Level Algorithm:*ComputeFixedPoint(Query)*

iteration indices in its head bound, and if $p(O)$ is true in the current iteration, the table entry will contain the iteration indices. $p(O)$ is known if it can be derived from the rules, calling it directly. Alternately, $p(O)$ is true if $O \in P$ was entailed by the the ontology in the last inner iteration step, as determined by the call $allModelsEntails/3$, which checks for the entailment of $O \in P$ in the previous inner iteration. In both cases, care must be taken so that it is guaranteed that if $\neg A$ holds, then $not\ A$ holds as well. In Definition 3 this is guaranteed by considering the addition of $\mathbf{not}NP$ in bodies of rules with head P in one of the alternating operators. Identically, when we try to derive $known(A, OutIter, InIter)$ and the iteration $OutIter$ is even (i.e. corresponding to a step where $\Gamma'_{\mathcal{K}}$, rather than $\Gamma_{\mathcal{K}}$, is being applied) , we further check if the ontology derived $\neg A$ in the last set. If so, then $known(A, OutIter, InIter)$ fails. This restriction is imposed by the predicate $no\_prev\_neg/3$:

```
known(A):-
     computeFixedPoint(A), get_object_iter(A,OutIter,InIter),
     known(A,OutIter,InIter).

:- table known/3.
known(A,OutIter,InIter):-
  (    call(A),
    ;
      InIter > 0, LastIter is InIter - 1,
      allModelsEntails(A,OutIter,LastIter) ),
  ( OutIter mod 2 =:= 1 -> true;
    no_prev_neg(A,OutIter, LastIter) ).

no_prev_neg(_A,_OutIter, LastIter) :- LastIter < 0,!.
no_prev_neg(A,OutIter, LastIter) :-
  tnot(allModelsEntails(not(A),OutIter, LastIter)).
```

Fig. 2. Prolog Implementation of $\mathbf{K}$ for Class Properties

On the other hand, the predicate $dlnot(A)$ which uses closed world assumption, succeeds if A fails. As discussed in Example 4, the evaluation of $dlnot/2$ must take into account the result of the previous *outer* iteration. Accordingly, in Figure 3 the call $dlnot(A)$ with $A = p(O)$ gets the current iteration for O, and immediately calls $dlnot/2$. The second clause of $dlnot/2$ simply finds the index of the fixed point of the previous outer iteration, and determines whether A was true in that fixed point. Since the call to $known/3$ in $tnot/1$ is tabled, none of the predicates for $\mathbf{not}$ need to be tabled themselves. As described before, each outer iteration represents an iteration in T and TU sets of Definition 7 for $MKNF_{WFS}$. As a result, T sets are monotonically increasing whilst TU sets are monotonically decreasing. To assure that the first TU set is the largest set, we compel all calls to $dlnot/1$s to succeed in the first outer iteration, as represented by the first clause of $dlnot/2$.

Ontology Component The tabled predicate *allModelsEntails/3* provides the interface to CDF's tableau theorem prover (Figure 4). It is called with an atom or its negation

```
dlnot(A):-
     computeFixedPoint(A),
     get_object_iter(A,OutIter,_InIter), dlnot(A,OutIter).

dlnot(_A,0):- !.
dlnot(A,OutIter):-
        LastIter is OutIter - 1, get_final_iter(A,LastIter,
            FinIter),
        tnot(known(A,LastIter,FinIter)).
```

Fig. 3. Implementation of **not** for Class Properties

and with the indices of its outer and inner iterations both bound. The predicate converts the atomic form of a proposition to one used by CDF. It translates a 1-ary DL-atom representing an individual's class membership to the CDF predicate $isa/2$, a 2-ary DL-atom representing an individual's role to the CDF predicate $hasAttr/3$ (see Section 3). In addition, if $Atom$ is a 2-ary role, the target individual may be added to the fixed point set of individuals.

```
:- table allModelsEntails/3.
allModelsEntails(not(Atom),_OutIter,_InIter):- !,
        /* transform Atom to CDF to an object identifier and
             class expression*/
        /* add individuals to current fixed point list */
        (rec_allModelsEntails(Id,CE) -> fail ; true).
allModelsEntails(Atom,_OutIter,_InIter):-
        /* transform Atom to CDF to an object identifier and
             class expression*/
        /* add individuals to current fixed point list */
        (rec_allModelsEntails(Id,not(CE)) -> fail ; true).
```

Fig. 4. Prolog Clauses for $allmodelsEntails/3$

The tableau prover, called by $rec\_allModelsEntails/2$, ensures that it obtains all information inferred by the rules during the previous inner iteration, in accordance with Definition 8. This is addressed via the CDF intensional rules. In general, the architecture of a CDF instance can be divided into two parts – extensional facts and intensional predicates. Extensional facts define CDF classes and roles as simple Prolog facts; intensional rules allow classes and roles to be defined by Prolog rules which are outside of the $MKNF_{WFS}$ semantics. In our case, the intensional rules support a programming trick to check rule results from a previous iteration. As shown in Figure 5 they directly check the $known/3$ table for a previous iteration using the predicate *lastKnown/1* (not shown). If roles or classes are uninstantiated in the call from the tableau prover, all defined roles and classes for the individual are instantiated, and called using *lastKnown/1* against the last iteration of the rules.

Discussion As described, *CDF-Rules* implements query answering to hybrid MKNF knowledge bases, and tries to reduce the amount of relevance required in the fixed point operation. Relevance is a critical concept for query answering in practical systems, however a poorly designed ontology or rules component can work against one

```
isa_int(oid(Obj,NS),cid(Class,NS1)):-
  ground(Obj),ground(Class),!,
  Call =.. [Class,Obj], lastKnown(Call).
isa_int(oid(Obj,NS),cid(Class,NS)):-
  ground(Obj),var(Class),!,
  definedClass(Call,Class,Obj), lastKnown(Call).

hasAttr_int(oid(Obj1,NS),rid(Role,NS1),oid(Obj2,NS2)):-
    ground(Obj1), ground(Obj2), ground(Role),!,
    Call =.. [Role,Obj1,Obj2],
    last_known(Call).
hasAttr_int(oid(Obj1,NS),rid(Role,NS1),oid(Obj2,NS2)):-
    ground(Obj1), ground(Obj2), var(Role),!,
    definedRole(Call,Role,Obj1,Obj2),
    last_known(Call).
```

Fig. 5. Callbacks from the ontology component to the rules component

another if numerous individuals depend on one another through DL roles. In such a case the relevance properties of our approach will be less powerful; however in such a case, a simple query to an ontology about an individual will be inefficient in itself. The approach of *CDF-Rules* cannot solve such problems; but it can make query answering as relevant as the underlying ontology allows. Optimizations of the described approach are possible. First is to designate a set of atoms whose value is defined *only* in the ontology: such atoms would require tableau proving, but could avoid the fixed point check of $computeFixedPoint/1$. Within $computeFixedPoint/1$ another optimization would be to maintain dependencies among individuals. Intuitively, if individual I_1 depended on individual I_2 but not the reverse, a fixed point for I_2 could be determined before that of I_1. However, these optimizations are fairly straightforward elaborations of *CDF-Rules* as presented.

5 Conclusions

In this paper we described the implementation of a query-driven system, *CDF-Rules*, for hybrid knowledge bases combining both (non-monotonic) rules and a (monotonic) ontology. The system answers queries according to *MKNF*$_{WFS}$ [7] and, as such, is also sound w.r.t. the semantics defined in [10] for Hybrid MKNF knowledge bases. The definition of *MKNF*$_{WFS}$ is parametric on a decidable description logic (in which the ontology is written), and it is worth noting that, as shown in [7], the complexity of reasoning in *MKNF*$_{WFS}$ is in the same class as that in the decidable description logic; a complexity result that is extended to a query-driven approach in [1]. In particular, if the description logic is tractable then reasoning in *MKNF*$_{WFS}$ is also tractable. Our implementation fixes the description logic part to CDF ontologies that, in its Type-1 version, supports $\mathcal{ALCQ}$ description logic. CDF Type-0 ontologies are simpler, and tractable and, when using Type-0 ontologies only our implementation exhibits a polynomial complexity behavior. This fact derives from the usage of tabling mechanisms, as defined in SLG resolution and implemented in XSB Prolog, though the proof of such is beyond the scope of this paper. For space reasons it was impossible to include here a proof of correctness of the implementation by relating it to the *MKNF*$_{WFS}$ tabling framework of [1], SLG($\mathcal{O}$): however this can be done by formally relating the iterative fixed point in Section 4 to the ORACLE RESOLUTION operation of SLG($\mathcal{O}$). We also omit here comparisons between MKNF and other proposals for combining rules and ontologies, as we focus on the implementation rather than on the definition of a semantics. For a survey on these proposals, see [6], and [10,7] for comparisons to MKNF.

CDF-Rules serves as a proof-of-concept for querying *MKNF*$_{WFS}$ knowledge bases. As discussed, XSB and tractable CDF ontologies have been used extensively in commercial semantic web applications; the creation of *CDF-Rules* is a step toward understanding whether and how *MKNF*$_{WFS}$ can be used in such applications. As XSB is multi-threaded, *CDF-Rules* can be extended to a *MKNF*$_{WFS}$ server in a fairly straightforward manner. Since XSB supports CLP, further experiments involve representing temporal or spatial information in a hybrid of ontology, rules, and rule-based constraints. In addition, since the implementation of Flora-2 [15] and Silk are both based on XSB, *CDF-Rules* forms a basis for experimenting with *MKNF*$_{WFS}$ on these systems.

References

1. J. J. Alferes, M. Knorr, and T. Swift. Queries to hybrid mknf knowledge bases through oracular tabling. In *The Semantic Web - ISWC 2009*, volume 5823 of *LNCS*. Springer, 2009.
2. W. Chen and D. S. Warren. Tabled Evaluation with Delaying for General Logic Programs. *Journal of the ACM*, 43(1):20–74, January 1996.
3. W. Drabent and J. Małuszynski. Well-founded semantics for hybrid rules. In *RR2007*, pages 1–15. Springer, 2007.
4. T. Eiter, T. Lukasiewicz, R. Schindlauer, and H. Tompits. Well-founded semantics for description logic programs in the semantic web. In *RuleML'04*, pages 81–97, 2004.
5. M. Gelfond and V. Lifschitz. Logic programs with classical negation. In Warren and Szeredi, editors, *International Conference on Logic Programming*. MIT Press, 1990.
6. P. Hitzler and B. Parsia. Ontologies and rules. In S. Staab and R. Studer, editors, *Handbook on Ontologies*. Springer, 2 edition, 2009.
7. M. Knorr, J. J. Alferes, and P. Hitzler. A coherent well-founded model for hybrid mknf knowledge bases. In *Europ. Conf. on Artificial Intelligence*, pages 99–103. IOS Press, 2008.
8. V. Lifschitz. Nonmonotonic databases and epistemic queries. In *International Joint Conference on Artificial Intelligence*, pages 381–386, 1991.
9. B. Motik. *Reasoning in Description Logics using Resolution and Deductive Databases*. PhD thesis, University of Karlsruhe, 2006.
10. B. Motik and R. Rosati. A faithful integration of description logics with logic programming. In *International Joint Conference on Artificial Intelligence*, pages 477–482, 2007.
11. T. Swift. Deduction in ontologies via answer set programming. In *International Conference on Logic Programming and Non-Monotonic Reasoning*, pages 275–289, 2004.
12. T. Swift and D. S. Warren. *Cold Dead Fish: A System for Managing Ontologies*, 2003. Available via `http://xsb.sourceforge.net`.
13. A. van Gelder. The alternating fixpoint of logic programs with negation. In *Principles of Database Systems*, pages 1–10. ACM Press, 1989.
14. A. van Gelder, K. A. Ross, and J. S. Schlipf. Unfounded sets and well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.
15. G. Yang, M. Kifer, and C. Zhao. Flora-2: A rule-based knowledge representation and inference infrastructure for the semantic web. In *CoopIS/DOA/ODBASE*, pages 671–688, 2003.